大模型技术深入浅出：从基础概念到前沿模型架构

DeepSeek-V3 / R1 / V4 / V4.1 · GLM-5 · Kimi K2.5 / K3 技术报告导读

技术文档

2026 年 9 月

目录

大模型技术深入浅出：从基础概念到前沿模型架构

> 本文以七篇 2024 年末至 2026 年的旗舰大模型技术报告为线索——DeepSeek-V3 / R1 / V4 / V4.1、GLM-5、Kimi K2.5 / K3——从"什么是 token"讲起，一路走到"890 字节/token 的 KV cache"与"分位数负载均衡"这样的前沿细节。
>
> 全文坚持一个写法：每个概念按**四层递进**展开—— **第一层「大白话」**用具体生活场景建立直觉；**第二层「慢慢推导」**用最小数值例子手把手算一遍；**第三层「正式定义」**给出术语、公式与维度标注；**第四层「论文实战」**讲它在七篇论文里的真实用法与数字。小白读前两层就能跟上，专家直接跳到后两层也不会失望。

如何阅读本文：三条路线

本文不是七篇论文的缩写，而是一条从"零基础"到"能独立读技术报告"的完整路径。按你的背景选一条路线（也可以三条都走，层层加深）：

路线 A：纯小白路线（约 3 小时）。 只读每节的「大白话」「慢慢推导」和各章末尾的「小白 FAQ」，跳过所有公式。读完你会知道：大模型为什么贵、为什么需要几千张显卡、为什么上下文越长越慢、2026 年的模型比 2024 年强在哪。

路线 B：工程师路线（约 6 小时）。 通读第 1～8 章全部内容，重点看"慢慢推导"里的数值例子和"论文实战"里的工程参数（并行度、显存账、量化粒度）。第 9 章可按需查阅。读完你应该能手算一个模型的 KV cache 大小、能向同事讲清为什么 decode 是带宽瓶颈。

路线 C：论文速读路线（约 2 小时）。 直接读第 10 章的七模型对比表和演进时间线，建立全局地图后回到第 9 章逐篇精读，遇到不熟的概念回查第 2～8 章对应小节。适合准备组会报告或面试的读者。

三条路线不是互斥的：很多读者的真实路径是先走 A 建立信心，几周后带着实际问题回来走 B，最后为了引文献走 C。本文的四层递进结构正是为这种"同一概念多次回访、每次更深一层"的读法设计的——第一次读懂直觉，第二次看懂推导，第三次能用公式自己复算论文里的数字，才算真正读完。

三个阅读约定：

1. **英文术语首次出现时给出中文解释**，如"张量并行（Tensor Parallelism, TP）"，后文直接用缩写。
2. **所有插图均提取自原论文 PDF**，每张图都配"怎么看这张图"的中文导读，先看导读再看图，能省下一半的理解时间。
3. **数字以论文笔记为准**：如"671B 总参、37B 激活"指 DeepSeek-V3，B = 十亿（billion）参数。

---

1. 大模型是怎么工作的

> 本章回答三个最朴素的问题：模型读的是什么（token）、它的身体长什么样（Transformer）、它学习和工作的方式有什么不同（训练 vs 推理）。后面所有章节的优化——并行、量化、压缩——都是在给这三件事"省钱提速"，所以先把地基打牢。

1.1 Token：大模型的"字母表"

【大白话】 你去国外餐厅点菜，菜单上全是外文，你一个字都不认识。怎么办？手机里存一本"编号对照表"：每道菜一个编号，你报编号、服务员上菜。大模型就是这样——它眼里没有"人工智能"这四个字，只有一串编号。token 就是"编号单位"：可能是一个字、半个词、一个标点，取决于那本对照表（词表）怎么编。你说的每句话先被切成 token、换成编号喂给模型；模型算出一个新编号，再翻译回文字给你看。

【慢慢推导】 切分不是按字数来的。比如英文 "unbelievable" 可能被切成 `un`、`believ`、`able` 三个 token，而常见的 "the" 就是一个 token。中文里"大模型"可能是两个 token，"机器学习"可能是两个。一本 128K 大小的词表，意思是对照表里一共有约 12.8 万个编号。**算给你看**：如果一句话"我今天想吃火锅"被切成 `[我][今天][想][吃][火锅]` 共 5 个 token，模型读 1000 字的文章大约要处理 1500 个 token；而 DeepSeek-V3 预训练用了 **14.8T（14.8 万亿）token**——相当于把上面的句子连续读 3 万亿遍，或读完约 10 万亿字的书籍，这就是"训练量"的直观含义。

【正式定义】 主流模型用字节对编码（Byte Pair Encoding, BPE）或其字节级变体 BBPE 构建词表：从单字节出发，反复合并语料中最高频的相邻符号对，直到词表达标。词表大小是模型重要规格：DeepSeek-V3/V4 系列沿用约 **128K**（129,280），Kimi K3 扩大到 **160K**。词表越大，同样文本切出的 token 越少（省计算），但词嵌入表（embedding table）参数越多。

【论文实战】 token 是一切成本的基本单位：V3 预训练 14.8T token、V4/V4-Flash 分别 33T/32T、V4.1 用 45T 多模态 token、GLM-5 用 28.5T token、Kimi K2 用 15T 文本 token、K2.5 再续 15T 视觉-文本 token。所有"训练成本 = 每 token 计算量 × token 总数"的账都从这两个数出

发。**一个有用的直觉校准**：1T token 大约是 7000 亿汉字——相当于把全中国图书馆的藏书读几十遍；45T 的多模态 token 里还混着图片切片（视觉 token），这就是"多模态预训练"在账本上的样子。

1.2 Transformer 的直觉：一个"全员信息交换 + 私人加工"的循环

【大白话】 想象一个剧组围读剧本，31 个演员围坐一圈，每人手里只有自己的一句台词。第一轮：每个人发言（注意力，Attention），其他人边听边在笔记本上记下"这句和我有什么关系"；然后各自回房间琢磨（前馈网络，FFN），把笔记消化成更深的理解。第二轮再围读，这时每个人发言的内容已经是"消化过第一轮信息"的版本。如此重复几十轮——DeepSeek-V3 是 61 层，即 61 轮围读；Kimi K3 是 93 轮——最后一个人综合所有理解，投票猜出"剧本的下一句台词"。Transformer 的每一层，就是一轮"围读 + 琢磨"。

【慢慢推导】 一层里到底发生了什么？把注意力缩到最小：**3 个 token、每个用 4 维向量表示**（真实模型是 7168 维，原理一样）。假设三个 token 的向量为

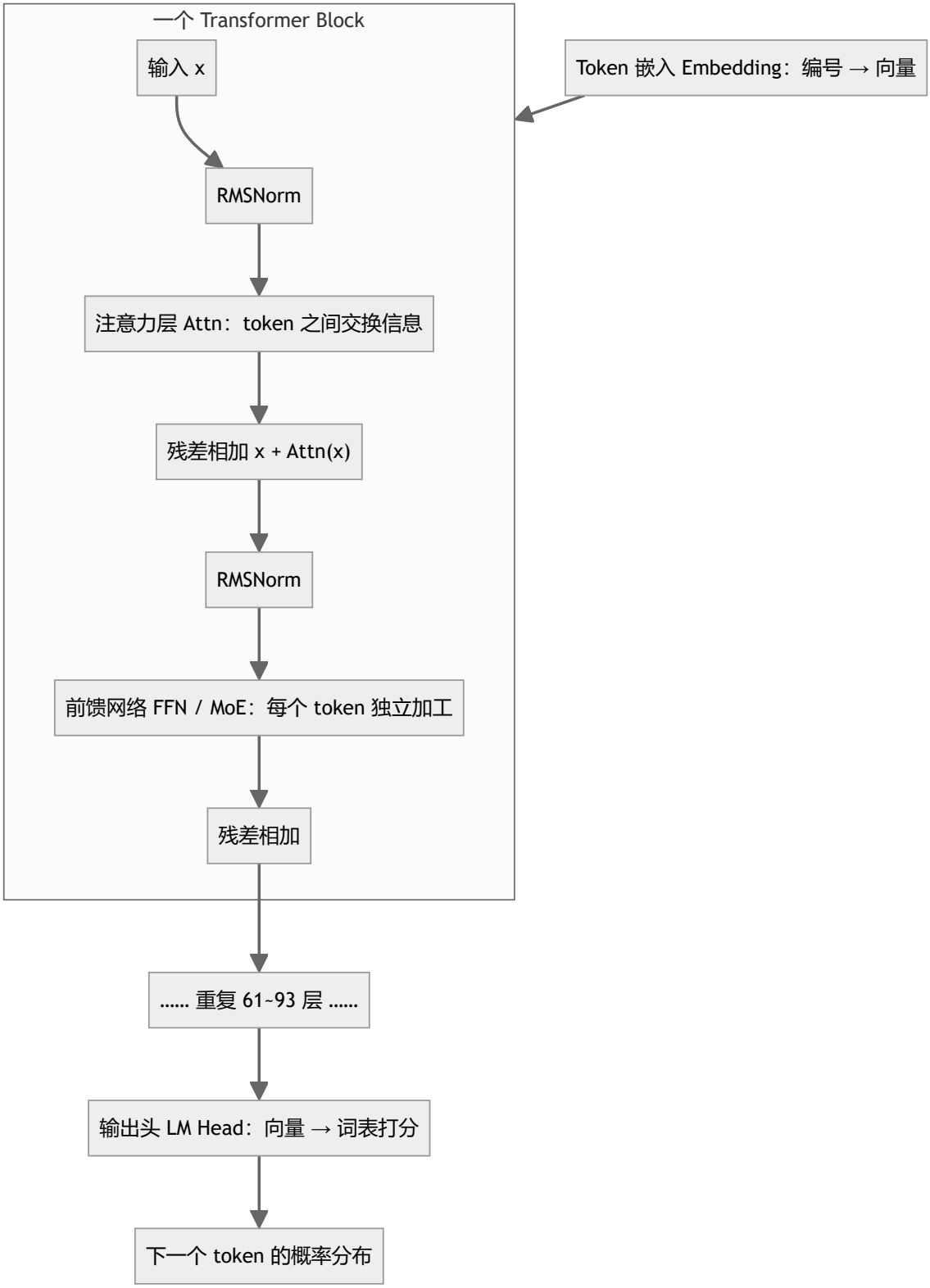
$$x_1 = (1,0,1,0), \quad x_2 = (0,1,0,1), \quad x_3 = (1,1,0,0)$$

每个 token 各自生成"查询"Q（我想找什么）、"键"K（我是什么）、"值"V（我的内容）三份副本。以 token 1 为例，假设它的查询 $q_1 = (1,0,0,0)$，三个键 $k_1 = (1,0,0,0)$、$k_2 = (0,1,0,0)$、$k_3 = (1,1,0,0)$。算"匹配分"就是点积：$q_1 \cdot k_1 = 1$，$q_1 \cdot k_2 = 0$，$q_1 \cdot k_3 = 1$。归一化（softmax）：$e^1 : e^0 : e^1 = 2.72 : 1 : 2.72$，归一后约 $(0.42, 0.16, 0.42)$。于是 token 1 的新表示 $= 0.42V_1 + 0.16V_2 + 0.42V_3$——**它对谁关注多，就"吸收"谁的内容多**。这就是注意力的全部本质：一个可学习的加权平均。FFN 则是对这个新表示做一次"私人加工"（两层全连接），不和别的 token 交流。

【正式定义】 标准 Transformer 块：

$$x' = x + \mathrm{Attn}(\mathrm{RMSNorm}(x)), \qquad y = x' + \mathrm{FFN}(\mathrm{RMSNorm}(x'))$$

注意力计算为 $\mathrm{Attention}(Q,K,V) = \mathrm{Softmax}(QK^T/\sqrt{d_h})V$。残差连接（residual connection，$x + \cdot$ 的加号）保证梯度可穿过上百层；RMSNorm 做归一化稳住数值范围。



【论文实战】 七篇论文全部沿用这个骨架，区别只在零件：DeepSeek-V3 用 MLA 注意力 + DeepSeekMoE 前馈层（61 层中除前 3 层外全部 MoE）；Kimi K3 用"3 个 KDA 线性注意力层 + 1 个 Gated MLA 层"的混合块；DeepSeek-V4.1 把 40 层劈成 20 层编码器 + 20 层解码器（CED）。骨架不变、零件全换——这是读懂所有论文的正确姿势。

【再算给你看：残差连接为什么不可或缺】 没有残差时，61 层的梯度要连乘 61 个雅可比矩阵，数值不是爆炸就是消失（各层奇异值若都约 1.1，$1.1^{61} \approx 340$；若都约 0.9，$0.9^{61} \approx 0.0016$）。有了 $x + \cdot$ 这条"高速公路"，梯度至少有一条恒等通路直达浅层，深网络才训得动。V4 的 mHC 和 K3 的 AttnRes 都是在这条高速公路上做文章——前者加宽车道但约束不超速（9.3 节），后者把"直通"改成"按需去历史各层挑货"（9.7 节）。

1.3 训练 vs 推理：学知识的阶段和用知识的阶段

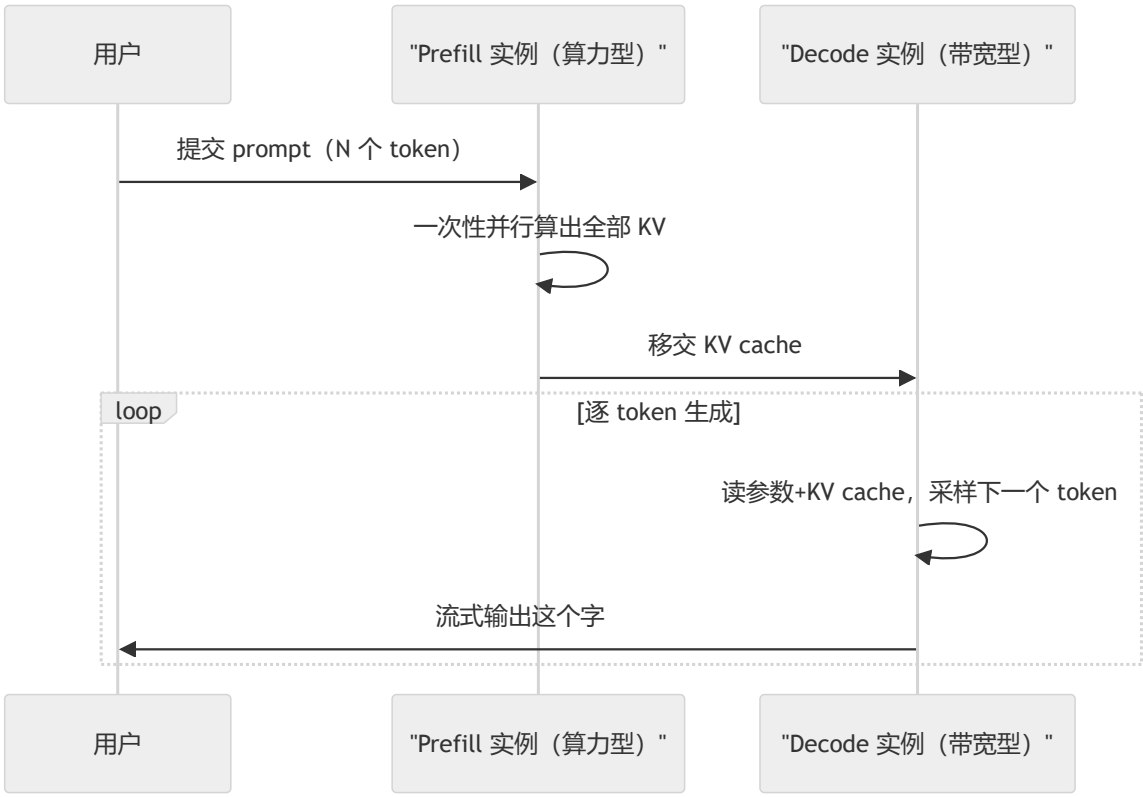
【大白话】 训练像学生的整个学生时代：做海量练习题，每道题做完立刻对答案、找差距、修正自己的知识体系（反向传播更新参数），代价巨大但只做几次。推理像考场答题：知识已定型，只能一个词一个词往外写，而且每写一个新词，都要回头把题目和自己已写的内容全部重读一遍。

【慢慢推导】 为什么"每写一词都要重读"？因为注意力需要所有历史 token 的 K/V。假设你已经写了 1000 个词，要写第 1001 个：它的查询要和前 1000 个词各自的键做点积。写第 1002 个时又要和前 1001 个点积。于是生成长度为 N 的回答，注意力总开销正比于 $1 + 2 + \cdots + N \approx N^2/2$——**上下文翻倍，注意力开销变四倍**，这就是"长上下文贵"的数学根源，也是第 5、6 章全部技术的靶子。**算给你看**：生成一篇 1 万 token 的长回答，注意力要做约 $10^8/2 = 5000$ 万对点积；若每对点积是 576 维（MLA decode 形态），仅点积就是约 $5 \times 10^7 \times 576 \times 2 \approx 576$ 亿次乘加——还没有算 61 层都要各做一遍。乘以 61 层后是 3.5 万亿次运算量级，与"每次都要读全部参数"的开销开始打平——这就是长上下文 decode 的双重瓶颈（既要搬参数，又要扫 cache）。

推理因此分成两个阶段：

- **Prefill（预填充）**：一次性并行处理用户输入的全部 N 个 token，是矩阵乘矩阵的大计算，GPU 算力吃得满——**计算密集型**。
- **Decode（解码）**：每步只生成 1 个新 token，但每次都要把全部模型参数和整个 KV cache 从显存读一遍——**带宽密集型**，GPU 大部分时间在等数据。

【论文实战】 两个阶段需求相反，业界普遍做 **PD 分离（Prefill-Decode 分离）**：DeepSeek-V3 部署时明确分离两者，decode 侧用"冗余专家"部署做负载均衡；GLM-5 的 slime 框架同样 PD 分离；DeepSeek-V4.1 用 CED 架构把 prefill 计算直接减半（9.6 节细讲）。



1.4 参数规模、激活参数与上下文长度：读懂模型规格表

【大白话】 规格表上的三个数字对应三本账：总参数量是"图书馆藏书总量"（模型有多大、占多少显存）；激活参数是"写一篇文章实际翻几本书"（每算一个 token 花多少算力）；上下文长度是"书桌上能同时摊开几页"（能看多长、KV cache 要多少）。

【慢慢推导】 以 DeepSeek-V3 为例：总参 671B、激活 37B，意味着每个 token 只动用 $37/671 \approx 5.5\%$ 的参数，稀疏度约 18 倍。推理时仅参数本体（BF16，每参数 2 字节）就要 $671 \times 2 \approx 1.34$ TB 显存——约 17 张 80GB 的 H800 才装得下模型本身，还没算 KV cache。Kimi K3 的稀疏度约 $2780/104 \approx 27$ 倍，V4.1 的 MoE 部分达 56 倍（896 选 16）：**总参一路膨胀、激活尽量按住**，是这两年最坚定的趋势。

【论文实战】 从 V3 的 128K 上下文到 V4/V4.1/K3 的 1M，两年涨近 8 倍，而每 token 推理成本不升反降——这就是第 5、6 章所有"压缩"技术的总成果。

【小白 FAQ】

- **Q：token 和字/词是一回事吗？** 不是。一个 token 约等于 0.6～0.8 个汉字、0.75 个英文单词，具体取决于词表。API 计费按 token 算，所以同样一段话不同模型价格不同。
- **Q：层数越多模型越聪明吗？** 不一定。层数（深度）、维度（宽度）、参数量、数据量共同决定能力，且要匹配算力预算。GLM-5 甚至故意"减层加宽"（80 层）来省专家并行

通信。

- **Q：为什么生成回答是一个字一个字蹦出来的？** 因为 decode 本质就是逐 token 串行的，第 $n+1$ 个词必须等第 n 个写完才能算。投机解码（第 8.1 节）能部分缓解。

---

2. 计算、内存、通信：大模型的三大瓶颈

上一章我们知道了模型"是什么"；这一章回答"为什么它这么贵"。所有前沿技术——FP8 训练、KV cache 压缩、并行策略——都可以归类为在和三个敌人作战：算不过来（计算）、装不下（内存）、传不动（通信）。

2.1 FLOPs：计算量怎么算

【大白话】 装修队报价按"工时"算：砌一块砖算一次工。FLOPs（浮点运算次数）就是大模型世界的"工时"——先估算总共要砌多少块砖，才知道要雇多少工人（GPU）、干多少天。

【慢慢推导】 一个参数在一个 token 上前向传播大约做 1 次乘加 = 2 FLOPs；反向传播要算"对输入的梯度"和"对参数的梯度"，约为前向的 2 倍 = 4 FLOPs。合计每参数每 token 6 FLOPs。**算给你看**：DeepSeek-V3 每 token 激活 37B 参数、训练 14.8T token：

$6 \times 3.7 \times 10^{10} \times 1.48 \times 10^{13} \approx 3.3 \times 10^{24}$ FLOPs

一张 H800 的 BF16 峰值算力约 989 TFLOPS，实际利用率按 40% 估约 400 TFLOPS = 4×10^{14} FLOPs/s，则需要 $3.3 \times 10^{24} / 4 \times 10^{14} \approx 8.2 \times 10^{9}$ 秒 ≈ 260 年单卡时间。摊到 2048 张卡上约 46 天——论文实测预训练 266.4 万 GPU 小时（2048 卡约 54 天），量级吻合，说明我们心算对了。

【正式定义】

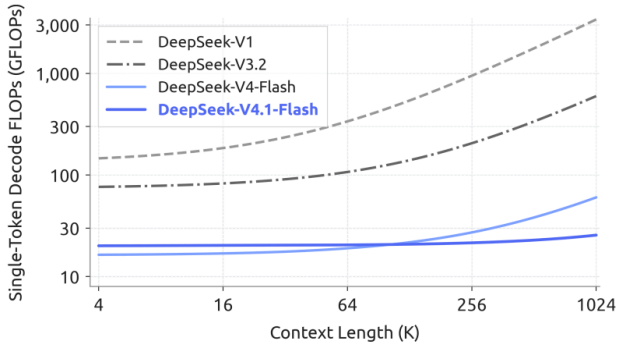
$C_{\text{train}} \approx 6 \times P_{\text{激活}} \times D$

D 为训练 token 数。推理侧单 token 约 $2 \times P_{\text{激活}}$ FLOPs（不计注意力随上下文增长的部分）。

【论文实战】 V3 总训练成本 2.788M H800 GPU 小时（预训练 2664K + 长上下文 119K + 后训练 5K），按 2 美元/GPU·时约 **557.6 万美元**；每 1T token 只需 180K GPU 小时。V4 在 1M 上下文下把单 token 推理 FLOPs 压到 V3.2 的 **27%**（3.7 倍下降）；V4.1-Flash 做到 decode FLOPs 随上下文几乎不变。

【推理侧也算给你看】 单 token 推理 FLOPs ≈ $2 \times P_{\text{激活}}$：V3 为 $2 \times 37B = 74$ GFLOPs；V4-Pro 为 $2 \times 49B = 98$ GFLOPs（注意力随上下文增长的部分另计）；V4.1-Flash decode 仅 $2 \times 16B = 32$ GFLOPs、prefill 更只有 $2 \times 8B = 16$ GFLOPs——**激活参数从 37B 降到 8B/16B，就是**

prefill 成本砍半再砍半的直接来源（CED 的功劳，9.4 节）。注意这个估算不含注意力项：dense 注意力每 token 还要扫全部历史 KV，上下文越长占比越大——这正是 V4/V4.1 把"随上下文增长的部分"逼近常数的原因（见本章开头配图）。



图：不同代 DeepSeek 模型单 token 解码 FLOPs 随上下文长度变化

怎么看这张图：横轴上下文长度、纵轴单 token 解码 FLOPs（BF16/FP8/FP4 按 1/0.5/0.25 加权折算）。传统注意力曲线随上下文线性上翘——读得越长、写每个字越贵；V4.1-Flash 几乎走平，这正是压缩+稀疏注意力的目标："读百万字的书，写每个字成本不变"。

2.2 显存都装什么：参数、梯度、优化器状态与 KV cache

【大白话】 训练像开一间工厂：机床（参数）占一份地，流水线在制品（激活值）占一份，质检记录（梯度）占一份，厂长的历史决策档案（优化器状态）占一份——而且档案比机床还大。推理像开阅览室：书（参数）加每位读者的读书笔记（KV cache），读者越多、读的书越厚，笔记越占地方。

【慢慢推导】 **算给你看**：用 AdamW 训一个 BF16 参数，每参数要存——FP32 主权重 4 B、梯度约 2～4 B、一阶动量 m 4 B（或 BF16 2 B）、二阶动量 v 4 B（或 2 B），合计 **12～16 B/参数**。一张 80GB 的 H800，裸训练只能装 80/14 ≈ 5.7B 参数。671B 的 V3 光"参数+训练状态"就需要约 $671 \times 14 \approx 9.4$ TB，除以 80GB ≈ 117 张卡起步——这还没算激活值和通信缓冲。这就是为什么需要 2048 张卡并精心设计分片（第 3 章）。

【正式定义】 推理期显存 = 参数（BF16 下 $2P$ 字节）+ KV cache（随上下文长度线性增长，第 6 章）+ 激活与中间缓冲。训练期另加梯度与优化器状态。

【论文实战】 优化器状态是省钱重点：V3 让优化器一、二阶矩用 BF16 存储；V4.1 的 Sinkhorn-balanced 更新"只需动量 buffer 却比 Adam 更好"，砍半优化器显存；Kimi K3 用统一激活管理，多数激活 block-wise FP8 量化后 offload 出显存。

【再算一笔：激活值】 上面只算了"长期档案"，训练时的"在制品"（每层前向的中间结果，反向传播要用）同样吓人。**算给你看**：V3 单序列 4K token、hidden 7168、61 层，只算每层输入激活（BF16）：$4096 \times 7168 \times 2B \times 61 \approx 3.6$ GB/序列——一个 batch 几十条序列就是上百 GB。这就

是为什么需要**重计算**（activation checkpointing：不存中间结果，反向时现场重算一遍，用算力换显存）和 CPU offload（暂时不用的激活搬去内存）。V3 正是靠这类极致优化把每卡显存压到不需要 TP 的水平。

2.3 通信与算术强度：为什么"算得快"不等于"跑得完"

【大白话】 GPU 像手艺惊人的厨师：颠勺 10 秒出一份菜。但外卖小哥（带宽）送一单要 5 分钟。餐厅产能由谁决定？骑手。大模型的"菜"是运算、"外卖"是数据搬运——很多场景下瓶颈不在算，而在搬。

【慢慢推导】 **算术强度（Arithmetic Intensity）** = 每搬运 1 字节数据能做多少次运算。**算给你看**：decode 生成 1 个 token，要读全部参数（V3 约 1.34TB 激活部分按 37B×2B = 74GB）但只做 $2 \times 37B = 74$ GFLOPs，算术强度 ≈ 1 FLOPs/B。H800 的平衡点约为算力/带宽 $\approx 989 \times 10^{12} / 3.35 \times 10^{12} \approx 295$ FLOPs/B。1 远低于 295——decode 有 99% 以上的时间在等数据，这就是"生成慢"的根源，也是**Roofline 模型**的含义：算术强度低于平衡点被带宽锁死（memory-bound），高于才被算力锁死（compute-bound）。

由此推出两个实用结论：decode 提速只有两条路——**少搬字节**（量化参数、压缩 KV cache）或**一次搬运服务更多 token**（增大 batch、投机解码一次验证多个草稿词）。

【论文实战】 带宽和延迟是两个敌人：H800 节点内 NVLink 约 160GB/s、节点间 InfiniBand 仅 50GB/s，差 3 倍多，所以热通信尽量留在节点内。所有通信优化无非三招：**少通信**（V3 node-limited routing 限制每 token 最多发往 4 个节点）、**藏通信**（DualPipe、wave 流水把通信垫在计算下）、**批量化**（合并小消息摊薄延迟）。V4 的结论是只要每字节通信配 6144 FLOPs 以上计算即可完全隐藏 EP 通信；V4.1 把 KV cache 压到 890 B/token，既是显存胜利也是 decode 速度胜利。

> **【小白 FAQ】**
>
> - **Q：为什么不一味买算力更强的卡？** 算力涨得比带宽快，"算等数据"的问题只会更严重。架构上省字节（压缩）往往比硬件上堆算力更划算。
> - **Q：训练 557 万美元是不是很贵？** 对比同期同水平模型动辄数千万至上亿美元的训练成本，V3 最大的震撼恰恰在于"便宜"——这来自第 3、4 章的系统与量化优化。

---

3. 并行计算全家桶：DP / TP / PP / EP / SP

> 第 2 章算出 V3 光训练状态就要 9.4TB 显存、单卡要算 260 年——答案只有一个：切开，摊到成千上万张卡上。本章五种并行方式各切一个维度。理解每种方式只需回答三个问题：**切什**

> 么、怎么切、通信在哪。

【总览大白话】 假设要把"读 100 本书并写读书报告"的活分给 4 个助理：可以每人各读 25 本用同一套模板（数据并行 DP）；可以把每本书撕成 4 份每人读所有书的同一部位再拼起来（张量并行 TP）；可以排成流水线，第 1 人读、第 2 人摘、第 3 人评、第 4 人排版（流水线并行 PP）；可以按科室分诊，心血管的病例都发给心内科助理（专家并行 EP）；书太厚时一人读一章、写报告前互相交换章节摘要（序列/上下文并行 SP/CP）。

3.1 数据并行（Data Parallelism, DP）

【大白话】 一个老师出题，全班 64 个学生各做一套不同练习题，做完互相核对"大家往哪个方向改"（梯度 all-reduce），保证全班朝同一方向进步。

【慢慢推导】 **算给你看（4 张卡各自拿什么）**：模型 10B 参数，batch 256 条数据。纯 DP：每张卡都拿**完整 10B 模型副本**，但数据各拿 64 条。算完各自得到一份梯度，4 卡把梯度对齐取平均（all-reduce，通信量 = 10B 参数 × 4 B = 40GB 量级，每步一次）。问题：每卡存完整副本太浪费——ZeRO（Zero Redundancy Optimizer）把优化器状态（ZeRO-1）、梯度（ZeRO-2）、参数（ZeRO-3）逐层切到各卡，谁要用临时 all-gather 聚拢。像合伙开图书馆：不再每人囤全套书，每人只藏一个学科的架子，要借再去取；省空间但多跑腿，ZeRO-1/2/3 是三档激进程度。**再算给你看（ZeRO 省多少）**：10B 参数、64 卡。纯 DP 每卡存全部 12~16 B/参数的状态 ≈ 140GB（装不下）；ZeRO-1 只分优化器状态：每卡优化器状态变成 $10B \times 8B/64 \approx 1.25$GB，加上参数+梯度共约 41GB；ZeRO-3 连参数梯度都分片，每卡仅约 2.2GB 常驻 + all-gather 通信开销。分片越狠、显存越省、通信越勤——这就是 ZeRO 档位的取舍本质。

【论文实战】 V3 用 ZeRO-1 DP；GLM-5 用 Pipeline ZeRO-2 gradient sharding（梯度跨 DP rank 分片，只保留 2 个 stage 的完整累积 buffer 双缓冲滚动）；K3 的 Muon 通信做零冗余化——all-gather 仅限本 rank 拥有的参数分片，消除峰值显存尖刺。**一个容易混淆的点**：DP 组数与 ZeRO 分片组数可以不同——例如 V3 是"64-way EP 之外再做 ZeRO-1 DP"，即在 EP 划好的组内部再按 DP 分数据、分优化器状态。实际部署时并行度写作乘积形式（如 V3 的 16PP × 64EP × 剩余 DP ≈ 2048 卡），读论文时看到并行配置，先验证"各维度乘积 = 总卡数"，就能快速定位每个维度负责什么。

3.2 张量并行（Tensor Parallelism, TP）

【大白话】 一张大饼一个人烙不过来，切成 8 瓣每人烙一瓣，最后拼成完整的饼。但每烙一层都要拼一次——**拼接动作极频繁**，所以 TP 一般只在同一台机器的 8 张卡内用（节点内 NVLink 160GB/s 才扛得住）。

【慢慢推导】 一个 4096 × 4096 的权重矩阵 $Y = XW$，切 4 路 TP：每卡拿 W 的 1/4（4096 × 1024 列），各算 XW_i 得 4096 × 1024 的部分结果，最后按列拼回 Y。前向拼一次、反向再散一次

——**每层两次集合通信**，是五种并行里通信最频繁的。**算给你看**：一个 61 层模型切 8 路 TP，每步仅 TP 通信就有 $61 \times 2 = 122$ 次 all-reduce；若每次通信的消息是 batch×seq×hidden 的激活（如 $4096 \times 7168 \times 2B \approx 59$MB），则每步 TP 流量约 $122 \times 59 \times 2 \approx 14$ GB——节点内 NVLink 160GB/s 约需 90ms，已经占了单步时间的可观部分；一旦跨节点走 50GB/s 的 IB，直接不可接受。这就是"TP 不出节点"的硬约束。

【论文实战】 通信太贵，DeepSeek-V3 明确**不用 TP**，靠极致显存优化（重计算、CPU offload）把每卡内存压到不需要 TP 的水平。

3.3 流水线并行（Pipeline Parallelism, PP）

【大白话】 工厂装配线：61 层的模型切成 16 个工位，半成品（激活值）在工位间传递。问题是工位会"等料空转"——这就是流水线气泡（bubble），优化的目标就是把空转时间填掉。

【慢慢推导】 **算给你看**：4 个 stage、8 个 micro-batch。最朴素的调度是先让所有 micro-batch 前向走完再全部反向（GPipe），气泡巨大；1F1B 调度（交替一次前向一次反向）把气泡压到 $(PP-1)(F+B)$，即"首尾各有一段只有 3 个 stage 在干活"的时间。DeepSeek-V3 论文给出三家对比：1F1B 气泡 $(PP-1)(F+B)$；ZeroBubble 的 ZB1P 为 $(PP-1)(F+B-2W)$（把反向拆成"对输入"和"对权重"两段插空）；**DualPipe** 为 $\frac{PP-1}{2}(F\&B+B-3W)$——气泡减半，代价是 2 倍参数副本和 PP+1 倍激活。

【论文实战】 V3 用 16-way PP；GLM-5 用交织式 PP（interleaved，把层拆得更碎交替放置，进一步减泡），并在交织 PP 下做 flexible MTP placement 均衡各 stage 显存。

3.4 专家并行（Expert Parallelism, EP）

【大白话】 医院分诊：导诊台（路由器）看完病历把病人（token）分到不同科室（专家），各科在不同楼层（不同节点），病人坐电梯过去（all-to-all 通信）、看完再送回来汇总。256 个专家摊在几十张卡上，每层都要"送病人、接病人"各一次。

【慢慢推导】 **算给你看（EP 通信量）**：V3 每个 token 发 8 个专家，hidden 7168。一次 dispatch 要发送 8 份 token 表示：$8 \times 7168 \times 2B \approx 115$ KB/token；一个 100 万 token 的 batch 就是约 115GB 的跨卡流量——61 层每层两次（dispatch + combine），总流量以 TB 计。V3 的两级转发设计把这笔钱省下来：token 先经 IB（50GB/s）只发到目标节点的"接待卡"，再经 NVLink（160GB/s）节点内转发到目标专家所在卡；配合 node-limited routing（最多 4 节点），每 token 实际通信成本等价于最多 13 个专家而非理论最坏值。V4 的 wave 流水进一步证明：只要每字节通信配 6144 FLOPs 以上计算（V4-Pro 每 token-expert 是 $6 \cdot h \cdot d$ FLOPs 对 3·h 字节 FP8 dispatch + BF16 combine），通信可被完全隐藏，理论加速 1.92 倍（对比 Comet 方案的 1.42 倍）。

【澄清一个常见误解】 EP 和 DP 看起来都"各卡算各的"，本质不同：DP 每卡做**同一件工作**（同一模型、不同数据），卡间交换梯度，可以攒到一步结束再一次通信；EP 每卡做**不同工作**

（不同专家、不同 token），卡间交换的是数据本体，通信卡在每层 MoE 中间，躲不掉只能藏。2026 年的系统创新几乎全围绕 EP 展开，原因在此。

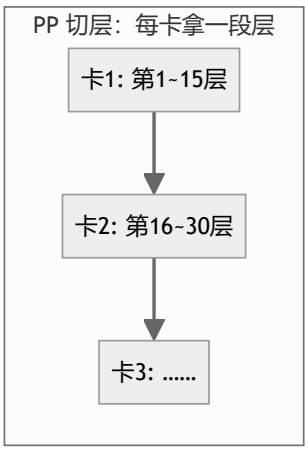
【论文实战】 V3 用 64-way EP 跨 8 节点，并把 all-to-all kernel 与网络拓扑协同设计：token 先经 IB 到目标节点同序号 GPU、再经 NVLink 转发，两种通信完全重叠——每 token 选 8 个专家但通信成本等价于最多 13 个（4 节点 × 3.2 专家/节点）。V4 做 wave 级流水：当前 wave 计算、下一 wave 收 token、已完成专家发结果三路并发，理论加速 1.92 倍。K3 开源 MoonEP：每 rank 恰好收 $S \times K$ 个 token、通信 buffer 固定 $S \times K$（zero-copy）、静态形状无 host 同步。

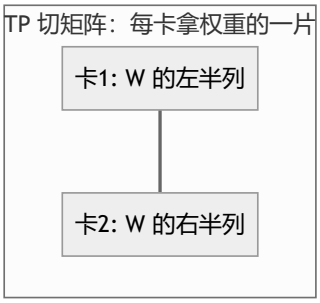
3.5 序列/上下文并行（Sequence/Context Parallelism, SP/CP）

【大白话】 多人合读一部长篇小说，每人读一章；写读后感时要引用别人章节的句子，于是开会互相交换章节笔记（KV）。

【慢慢推导】 算给你看（**CP** 怎么分）：1M token 序列切 8 卡，每卡本地存 125K token 的 KV。算注意力时，第 i 卡的 query 需要看到前 i 卡所有的 K/V（因果性）——两种经典方案：all-gather（把别人的 KV 全收过来，简单但显存爆）或环形传递（ring attention，KV 在卡间转圈，每圈算一部分注意力，显存省但延迟高）。压缩注意力让这笔账变了：V4 的 CSA 里 rank i 只需把最后 m 条**未压缩** KV 发给 rank $i+1$ 本地压缩，再 all-gather 压缩后的条目——通信量从"每 token 一份 KV"降到"每 4 token 一份压缩 KV"，CP 的通信预算直接 ÷4。

【论文实战】 V4 为压缩注意力设计专门 CP：rank i 把最后 m 条未压缩 KV 发给 rank $i+1$ 本地压缩（$s/m+1$ 条），再 all-gather 收集 + fused select-and-pad 重排。K3 的 **KCP** 更精巧：delta rule 循环状态不能直接相加，每段分解为"累积转移矩阵 M + 本地从零状态 $\hat{S}$"，按 $S = \hat{S} + \prod M \cdot S_{in}$ 精确复合。





3.6 组合：3D 并行与 DualPipe

并行方式	切什么	主要通信	通信频率	典型规模	论文实例

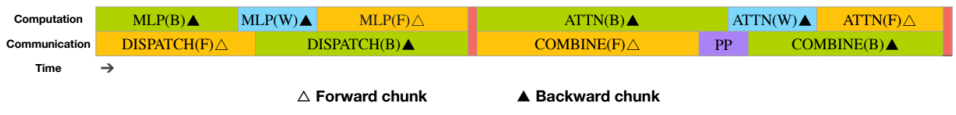
DP (+ZeRO)	数据 batch	梯度 all-reduce	每步一次	任意	V3: ZeRO-1；GLM-5: Pipeline ZeRO-2
TP	层内矩阵	层内 all-reduce	每层两次	≤8（节点内）	V3 明确不用
PP	层	激活点对点	每 micro-batch	16+	V3: 16-way；GLM-5: 交织 PP
EP	MoE 专家	all-to-all	每层两次	64+	V3: 64-way 跨 8 节点；K3: MoonEP
SP/CP	序列长度	KV all-gather	每注意力层	视上下文	V4 压缩注意力 CP；K3 KCP

【大白话】 DualPipe 的理解钥匙是"拉链"：前向 micro-batch 从左往右流，反向从右往左流，两列牙齿交错咬合；每个 chunk 拆成 attention / all-to-all dispatch / MLP / all-to-all combine 四段，让一张卡"算 attention 的时候顺便把上一段的数据发出去"。通信不是消失了，而是被计算挡住了。

【慢慢推导】 算给你看（气泡到底省多少）：设前向耗时 F、反向 B、对权重的反向 W（B 中含 W）。16-way PP（PP=16）：1F1B 气泡 $= 15(F+B)$，若 $F=1, B=2$（反向约两倍于前向），气泡 = 45 个时间单位，而理想无泡的流水线主体约等于 micro-batch 数 ×（F+B）。DualPipe 气泡 $= \frac{15}{2}(F\&B+B-3W)$，取 $W \approx 0.67$ 时约为 $7.5 \times (3-2) \approx 7.5$——**气泡缩小约 6 倍**。代价记住两条：参数要存 2 份副本、激活多存 PP+1 倍。这就是为什么它只在 V3 这种显存已被极致优化、但通信带宽受限（H800 的 IB 只有 50GB/s）的集群上划算。

图：DualPipe 调度示例

怎么看这张图：时间轴甘特图，8 个 PP rank、20 个双向 micro-batch。每格是某 rank 某时刻做的事，黑框圈住的两格表示计算与通信重叠。读图时专找空白格——空白越少气泡越小。



图：前向/反向 chunk 的计算-通信重叠策略

怎么看这张图：橙色=前向、绿色=对输入的反向、蓝色=对权重的反向、紫色=PP 通信、红色=同步屏障。注意 Transformer 块边界故意不对齐——错位设计让 attention 计算正好盖住 all-to-all 通信的时间。V4 将其演进为 "DualPipe 1F1B overlapping" 变体。

> 【小白 FAQ】
>
> - **Q：TP 和 EP 能一起用吗？** 能，但很少同时开在热点层。TP 通信每层都要做、EP 的 all-to-all 也是每层，叠加会让通信爆炸。V3 的选择是 EP 做跨节点、节点内靠显存优化省掉 TP。
> - **Q：为什么不只用 DP 把 batch 开到无穷大？** DP 要求每卡装下完整模型（或 ZeRO-3 花更多通信换）；且 batch 有算法上的合理上限，不是越大越好。
> - **Q：流水线气泡能彻底消除吗？** 理论上不能为零（首尾 stage 总有空转），DualPipe 已是当时最优之一，代价是参数副本翻倍。

---

4. 量化：用更少的比特存一个数

> 第 2 章说 decode 是带宽瓶颈——参数每小一半，生成就快一倍。本章讲怎么把每个数字"压扁"而不压坏模型。

4.1 数值格式速览

【大白话】 记账：每笔支出精确到分（FP32）安全但账本厚；记到角（BF16）够日常；只记整数元（INT8）偶尔出错；FP4 相当于只记"几十/几百/几千"的量级——必须配合"分组记账"才不乱套。

【慢慢推导】 算给你看：**0.375 在 E4M3 里怎么表示。** FP8 E4M3 格式：1 位符号 + 4 位指数 + 3 位尾数，指数偏置 7。$0.375 = 1.5 \times 2^{-2}$：符号位 0；指数 $-2 + 7 = 5$ 即二进制 0101；尾数 1.5 的小数部分 0.5 → 3 位尾数 100。拼起来 `0 0101 100` = 0x2C。能精确表示！但 0.377 就要四舍五入到 0.375 或 0.40625——**3 位尾数意味着相邻可表示数之间差 12.5%**，这就是 FP8 会引入误差的直觉来源。再看 FP4 E2M1（1 符号+2 指数+1 尾数）：正数只能表示 0、0.5、1、1.5、2、3、4、6 共 8 个值——不给每个小块配缩放因子根本没法用。

格式	位宽	指数/尾数	特点与用途
FP32	32 bit	8E / 23M	训练"金标准"，master weights 常用
BF16	16 bit	8E / 7M	动态范围大（指数位同 FP32），预训练主流
FP16	16 bit	5E / 10M	精度高范围小，易溢出
FP8 (E4M3)	8 bit	4E / 3M	V3 首次在 671B 规模验证训练可用

FP4 (E2M1)	4 bit	2E / 1M	推理 KV cache / QAT 权重前沿
INT8	8 bit	整数	经典推理量化

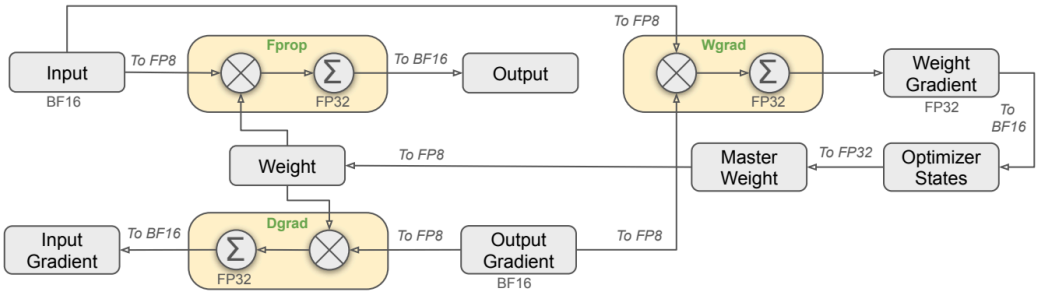
4.2 量化为什么可行

【大白话】 神经网络像橡皮泥做的秤：刻度差千分之一，称出来的重量几乎不变（对扰动鲁棒）。麻烦的是偶尔冒出的"超级大数"（outlier）——全班平均分 80，来一个 10000 分的，按最高分定刻度，其他人的成绩全被压成 0。解决方案：**分组定刻度**。

【慢慢推导】 算给你看（量化粒度）：一组激活 `[0.1, -0.3, 0.2, 50.0]` 用 INT8（范围 ±127）量化。per-tensor：缩放 = 50/127 ≈ 0.394，则 0.1 → 0（四舍五入 0.1/0.394 = 0.25），信息全丢。若按小块分组、前三个一组自己定缩放 0.3/127 ≈ 0.0024：0.1 → 42、-0.3 → -125，精度保住；50 单独一组。这就是"per-tensor < per-channel < group/block"粒度谱系的含义。DeepSeek-V3 用激活 **1×128 tile-wise**（每 token 每 128 通道一组 scale）、权重 **128×128 block-wise**，缩放因子在线取绝对值最大值（absmax）求得——这是 FP8 训练能稳定的关键。

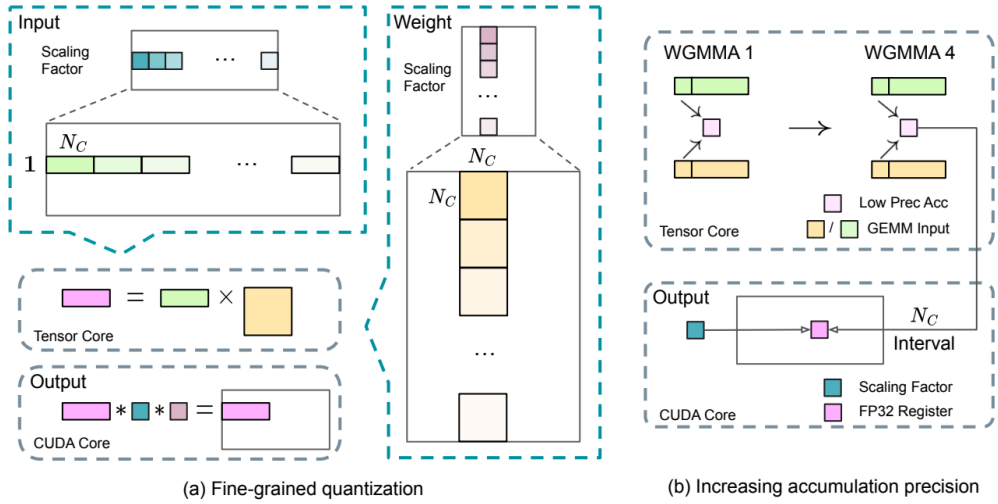
【论文实战】 V3 的完整精度布局：三类 GEMM（Fprop/Dgrad/Wgrad）全部 FP8；但 embedding、输出头、MoE 门控、归一化、attention 算子保留 BF16/FP32（"敏感器官"不动刀）；master weights、梯度、优化器状态高精度，优化器一、二阶矩 BF16。另一个坑：H800 的 FP8 GEMM 累加精度仅约 14 bit，V3 每算 128 个元素就提升到 CUDA core 用 FP32 累加一次，把误差拦住。实测相对 BF16 基线损失误差 **< 0.25%**。

【再算给你看：Wgrad 全 FP8 为什么是关键一步】 三类 GEMM 里 Wgrad（权重梯度）最难 FP8 化：它对精度最敏感，但若不 FP8，反向时激活就得用 BF16 缓存——显存翻倍。V3 把 Wgrad 也 FP8 化后，**激活可以直接以 FP8 缓存**，训练显存又省一刀。这就是"量化框架"和"量化某个算子"的区别：前者要让数据流上的每一环都自洽，一环退回高精度，上下游的存储格式都得跟着妥协。



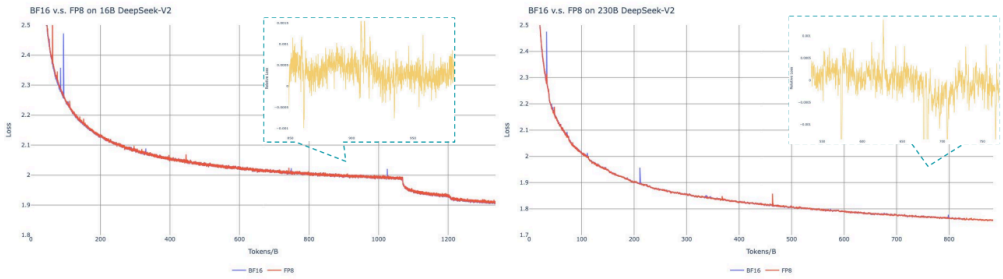
图：FP8 混合精度训练框架

怎么看这张图：以一个 Linear 算子为例，追踪前向、输入梯度、权重梯度三条 GEMM 通路的精度安排。读图要点：哪些箭头标 FP8（大部分），哪些保留高精度（外围敏感组件）。



图：细粒度量化：按 tile/block 缩放

怎么看这张图：(a) 演示为什么需要细粒度——outlier 会把共用 scale 撑爆；分成 1×128 小块各自定 scale 后，outlier 只影响自己那块。(b) 配套的 FP32 定期累加流程。



图：BF16 与 FP8 训练 loss 曲线对比

怎么看这张图：16B 与 230B 两个规模上 FP8 与 BF16 的 loss 曲线几乎重合，相对误差稳定在千分之一量级。这是"FP8 能用于超大模型预训练"的直接证据。

4.3 各论文的量化实践

- **DeepSeek-V3**：首次 671B 级 FP8 混合精度训练，统一 E4M3。
- **DeepSeek-V4**：KV cache 混合精度——RoPE 维度保 BF16、其余 FP8，cache 近减半；lightning indexer 的 attention 用 **FP4** 计算；后训练 **FP4 QAT（量化感知训练，MXFP4）**，且 FP4→FP8 反量化无损（E4M3 比 E2M1 多 2 位指数，能吸收 1×32 FP4 子块的细粒度 scale），整个 QAT 复用 FP8 训练框架，反向用直通估计器（STE，Straight-Through Estimator：反向时假装量化是恒等操作让梯度流过去）。
- **DeepSeek-V4.1**：FP4 推到主 **KV cache**——E2M1 + 每 16 通道一个 E4M3 scale。论证去掉全局 scale 无损：512 通道 latent 经 RMSNorm（权重幅度≈1）后 L2 范数 $\leq \sqrt{512}$、绝对值 ≤

22.6，实测峰值约 10，而格式上限 $448 \times 6 = 2688$，余量两百多倍。SWA 的 KV 对量化敏感，保留 FP8。

- **Kimi K3**：QAT 从 SFT 阶段起全程开启，MXFP4 权重 + MXFP8 激活；训练激活多数 block-wise FP8。
- **GLM-5**：RL rollout 侧 FP8 推理降低单 token 延迟。

> 【小白 FAQ】
>
> - **Q：FP8 训练会不会掉点？** V3 的实测是不会（损失误差 < 0.25%），前提是细粒度缩放 + FP32 定期累加 + 敏感组件保留高精度三件套齐备。裸上 FP8 必然掉点甚至发散。
> - **Q：FP4 都能用，为什么不直接上 1-bit？** 位宽越低，需要的"补救工程"越重（更细的分组、QAT 重训、敏感组件豁免）。FP4 目前只敢用在 KV cache 和部分权重上，且都配 QAT；1-bit 的误差已超过现有补救手段的能力范围。
> - **Q：量化是训练技术还是推理技术？** 都是。推理量化省带宽提速；训练量化（FP8）省算力与显存。QAT 是两者的桥：训练时模拟低精度，让模型提前适应。

---

5. Attention 机制的演进：从 MHA 到线性注意力

> 第 1 章说过：生成长度 N 的回答，注意力开销正比于 N^2，而且每个历史 token 的 K/V 都得存着（KV cache）。当上下文冲向 100 万 token，这条定律就是灾难。本章讲的六代注意力变体，本质上都在回答同一个问题：**怎样少记、少看，又不变笨**。

5.1 MHA：标准的多头注意力

【大白话】 全班 n_h 个读者同时审阅同一份档案，每人关注不同侧面（一个盯语法、一个盯指代、一个盯时间线），最后把笔记汇总。问题是：每个人的档案副本（K、V）都要存档，档案室（KV cache）很快就爆了。

【正式定义】 每个 token 产生 Q、K、V，分 n_h 个头各自计算 $\text{Softmax}(QK^T/\sqrt{d_h})V$ 再拼接投影。推理时每 token 每层要存 $2 \times n_h \times d_h$ 个数。

【慢慢推导】 算给你看：一个 60 层、hidden 2048、16 头、头维 128 的模型（2048 维正是 16×128），BF16 存储：每 token KV cache = $2(K\text{和}V) \times 60\text{层} \times 16\text{头} \times 128\text{维} \times 2\text{字节} = 491,520$ 字节 ≈ **480 KB/token**。一段 10 万 token 的上下文就是 48GB——一张 80GB 的卡，装完模型只剩一点地方，光一个长对话的 cache 就塞不下。这就是为什么后面每一代改动都围绕"砍这个数"。

5.2 MQA 与 GQA：让头们共享档案

【大白话】 既然 16 个读者的档案高度重叠，不如共用：**MQA（Multi-Query Attention）** 是所有查询头共享一份 K/V（全班共用一本档案）；**GQA（Grouped-Query Attention）** 是每 g 个头共享一份（每 4 人小组共用一本）。省：cache 直接除以 n_h 或 n_h/g。代价：共享等于信息损失，MQA 掉点明显，GQA（如 8 组）是效果与成本的折中，成为 Llama 等模型的标配。

【慢慢推导】 算给你看：沿用 5.1 的 60 层、16 头、128 维模型，每 token cache 480 KB。MQA（共享成 1 份 K/V）：$2 \times 60 \times 1 \times 128 \times 2B = 30,720$ B ≈ **30 KB/token**，缩小 16 倍；GQA-8（8 组共享）：$2 \times 60 \times 8 \times 128 \times 2B = 245,760$ B ≈ **240 KB/token**，缩小 2 倍。注意一个不对称现象：查询 Q 不共享也不缓存（Q 只为当前 token 服务、用完即弃），被共享和缓存的永远是 K/V——所以这类技术的正式名称才叫"Multi-**Query**"而不是"Multi-Key"。这个不对称在第 5.3 节 MLA 的"cache 只存 latent"里还会再出现。

5.3 MLA：把 K、V 压成一个"摘要向量"

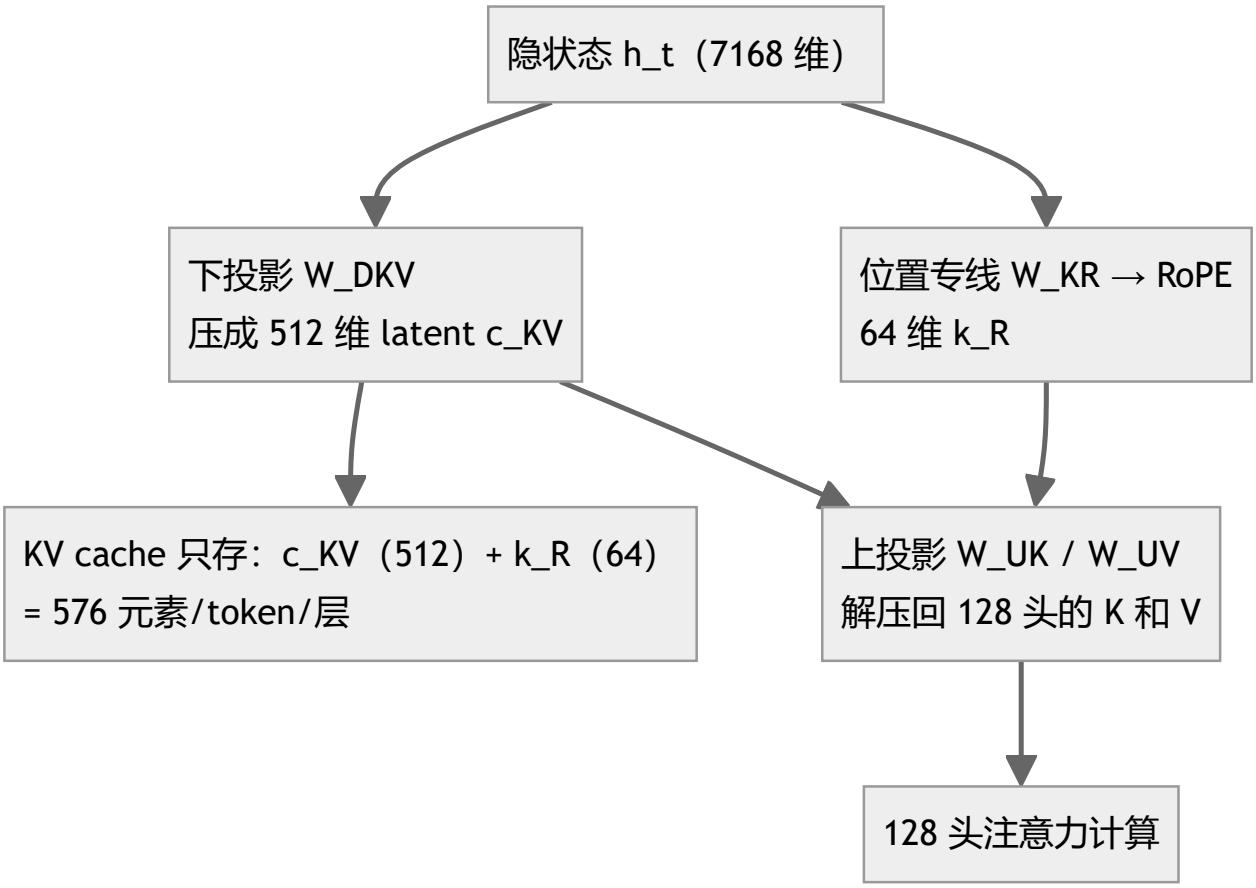
【大白话】 **MLA（Multi-head Latent Attention，多头潜在注意力）的思路换了方向：不共享档案，而是只存一份 512 字的会议纪要**（latent vector），将来谁要引用，现场根据纪要重建成他需要的格式。比共享聪明在哪？重建是带参数的"解压"，每个头解压出的内容各不相同——信息没有丢，只是换了存储形式。

【慢慢推导】 算给你看（**MLA 的维度账**）：V3 每层的 K/V 下投影矩阵 $W_{DKV} \in \mathbb{R}^{512\times 7168}$：把 7168 维的隐状态压成 512 维 latent c_t^{KV}，参数量 $512 \times 7168 \approx 3.7$M；上投影 W_{UK} 把 512 维解压回 128 头×128 维 = 16384 维的 K，参数量 $16384 \times 512 \approx 8.4$M。若不用 MLA，直接投影 K 需要 $7168 \times 16384 \approx 117$M 参数。**压缩反而省了参数**。推理时缓存从 MHA 的 $2 \times 128 \times 128 = 32768$ 元素/token/层，降到 $512 + 64 = 576$ 元素——**约 57 倍压缩**。那 64 是什么？是解耦的 RoPE 分支：位置编码不能压进 latent（压进去就没法把上投影吸收进相邻矩阵），所以单独留一条 64 维"位置专线" $k_t^R = \mathrm{RoPE}(W_{KR} h_t)$，与压缩部分拼接使用。

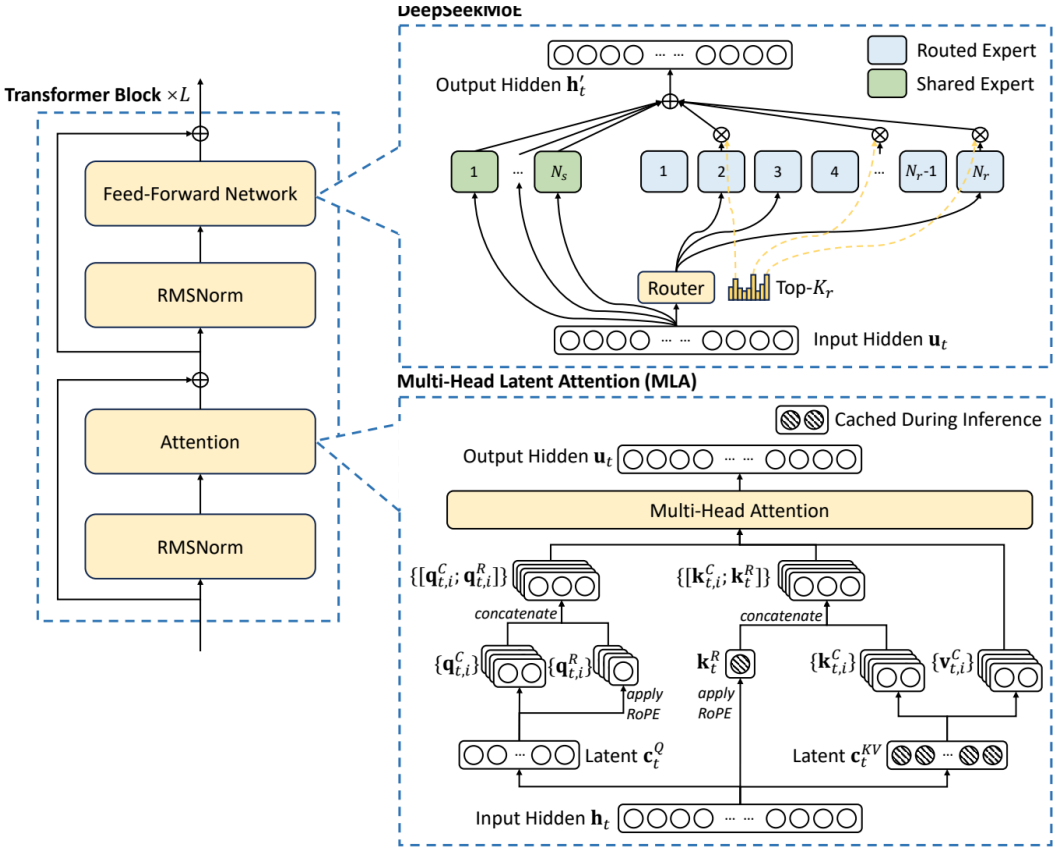
【正式定义】（V3 原文公式）

$$c_t^{KV} = W_{DKV} h_t \in \mathbb{R}^{512}, \quad k_{t,i}^C = W_{UK} c_t^{KV}, \quad v_t^C = W_{UV} c_t^{KV}, \quad k_{t,i} = [k_{t,i}^C;\, k_t^R]$$

注意力输出 $o_{t,i} = \sum_j \mathrm{Softmax}(q_{t,i}^T k_{j,i} / \sqrt{d_h + d_h^R}) v_{j,i}^C$。Query 同样低秩压缩（$d_c' = 1536$），省训练激活显存。推理时上投影矩阵可被吸收（matrix absorption），解压零额外成本。



【代价与插曲】 训练/prefill 时 MLA 仍是 MHA 形态的计算量（省的是推理 cache，不是训练算力）。GLM-5 还发现一个隐蔽问题：换用 Muon 优化器后，576 维 latent 的 MLA 追不上 2048 维 cache 的 GQA-8——病根不在结构容量，而在 Muon 对整个上投影矩阵统一正交化时，强迫各注意力头以相同尺度更新，抹平了头间异质性。解法：把 $W_{UQ}/W_{UK}/W_{UV}$ **按头拆成小矩阵分别正交化**（Muon Split），MLA 立即追平 GQA-8，且训练全程 logits 尺度稳定、无需任何 clipping；进一步把头维从 192 提到 256、头数减 1/3（**MLA-256**），训练计算与参数不变、decode 点积更便宜。这个插曲的教训：**架构不能脱离优化器、精度环境孤立评估——"架构 × 训练配方"才是真正被选择的对象**。



图：DeepSeek-V3 架构：MLA + DeepSeekMoE

怎么看这张图：左/中部是 MLA 数据流——注意 c_t^{KV} 这个"瓶颈"，所有 K/V 从这里解压，绿色小支路是解耦 RoPE key；右侧 MoE：共享专家 + Top-Kr 路由专家。这是理解后续所有 DeepSeek 模型的"母图"。

5.4 稀疏注意力：不是所有 token 都值得看

【大白话】 写论文时你不会把图书馆每本书都重读一遍——先查索引（indexer），找到最相关的几十页再精读。稀疏注意力就是这个流程：给历史 token 快速打分，只对 top-k 个做精细注意力。GLM-5 的实测给出了惊人的正当性：**长上下文中 90% 的注意力条目是冗余的**。

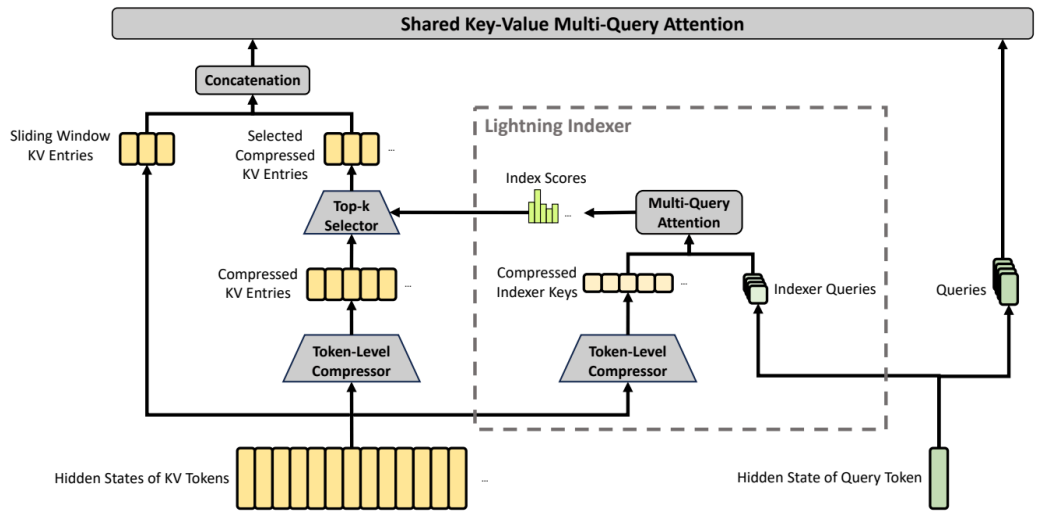
【论文实战】

- **DSA（DeepSeek Sparse Attention）**：轻量 **Lightning Indexer** 打分 + top-k 精细注意力。GLM-5 用它续训：dense warm-up 1000 步（14 序列 × 202,752 token，max LR 5e-3）+ sparse adaptation 仅 20B token（DeepSeek-V3.2 用了 943.7B，便宜了 47 倍），实现长序列 attention 计算降 1.5～2 倍、128K 下半价 GPU 成本，SFT loss 与 MLA 完全打平。工程细节：不用 indexer replay（k=2048 存索引太贵），改用**确定性 top-k 算子**消除训练与推理的 mismatch。

而非 512）+ 精细注意力（1024 次高维点积）。相比 dense 的 10^6 次高维点积，高维部分省了约 **1000 倍**——加上低维 indexer 的廉价粗筛，整体长序列注意力计算降 1.5～2 倍（GLM-5 实测），这就是"用便宜的索引换昂贵的注意力"的不等式。

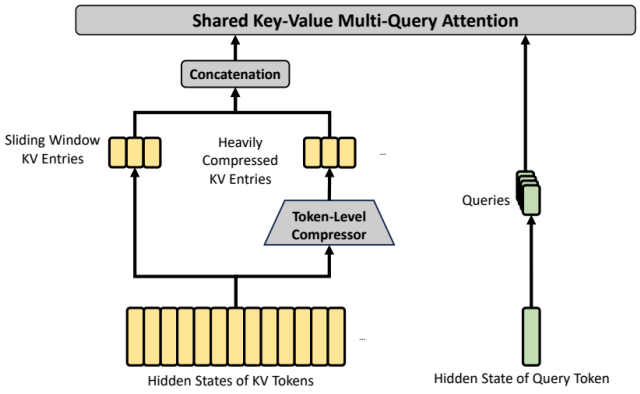
- **CSA（Compressed Sparse Attention，V4）**：先压再稀疏的两级漏斗——第一级把每 $m = 4$ 个 token 的 KV 压成 1 条（带压缩权重 Z 和可学习位置偏置 B 的 Softmax 加权合并，两路 C_a/C_b 错位重叠），条目数先除 4；第二级在压缩条目上跑 DSA（indexer 64 头 × 128 维，top-k 512/1024）。核心注意力用 **Shared-KV MQA**：每条压缩 KV 同时当 key 和 value、所有 query 头共享。输出侧 n_h×c 很大（64×512），先分 g 组投影到 $d_g = 1024$ 再汇总（Grouped Output Projection）。
- **HCA（Heavily Compressed Attention，V4）**：同一结构把压缩率推到 $m' = 128$、不做稀疏选择——"把 128 页压成 1 页摘要再看"。V4 让 CSA 与 HCA 层交错，外加 128 token 滑窗分支补足局部细节，再加每头一个可学习 attention sink（允许总注意力小于 1 甚至接近 0）。

【慢慢推导】算给你看（HCA 层的极致账本）：1M token 上下文经过 HCA（$m' = 128$）后只剩 $10^6/128 \approx 7,812$ 条 KV；再对比 dense 的 10^6 条——**条目数缩小 128 倍**。每条 512+64 维、FP8 存储约 576 B，HCA 层每 token 等效 cache $\approx 576/128 \approx 4.5$ B。即使所有层都是 HCA（实际是与 CSA 交错），1M 上下文的全局 KV 也就几十 MB 级——这就是为什么 V4 敢说"1M 下 KV cache 仅为 V3.2 的 1/9.5"。滑窗分支的存在则回答了"压缩会不会丢细节"：最近 128 个 token 保留原始 KV，局部依赖一条不丢。



图：CSA 核心结构

怎么看这张图：从左往右：每 token 原始 KV → 压缩器 4 条合 1 条 → Lightning Indexer 打分 top-k 圈出重要条目 → 选中条目 + 滑窗原始 KV 一起进注意力。"压缩"与"稀疏"两级漏斗各过滤一批。



图：HCA 核心结构

怎么看这张图：结构与 CSA 几乎相同，唯一区别是压缩率 $m' = 128 \gg 4$ 且不重叠、不做 top-k。配合滑窗分支弥补压缩块内的细节丢失。

5.5 线性注意力：KDA，把"记事本"换成"压缩饼干"

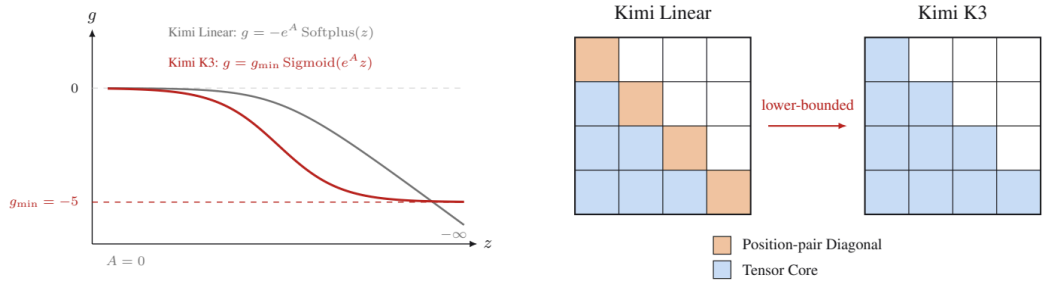
【大白话】 前面所有方案都在"怎么把越堆越高的档案柜塞小"；**KDA（Kimi Delta Attention）**直接不要档案柜——换成一块固定大小的白板：每来一个新 token，先按"逐通道遗忘速度"让旧字迹各自褪色，再把新内容写上去，写之前还会精准擦掉与旧内容冲突的部分（delta rule）。白板大小固定，一百万个 token 也不增加存储；代价是陈年老事会被磨掉——所以 K3 每 4 层插 1 层带档案柜的全局注意力（Gated MLA）兜底。

【正式定义】

$$S_t = (I - \beta_t k_t k_t^T)\,\mathrm{Diag}(\alpha_t)\,S_{t-1} + \beta_t k_t v_t^T, \qquad \tilde{o}_t = S_t^T q_t$$

$S \in \mathbb{R}^{d_k \times d_v}$ 是固定大小循环状态，$\alpha_t \in (0,1)^{d_k}$ 是逐通道遗忘门，β_t 是写入强度，$-\beta_t k_t k_t^T$ 表示"写入前先擦掉旧值"。q/k/v 经 ShortConv + Swish，q/k 加 L2Norm；输出门为全秩输入相关投影 $y_t = W_o[\mathrm{Sigmoid}(W_g x_t) \odot \mathrm{RMSNorm}(\tilde{o}_t)]$。

【慢慢推导】 算给你看（为什么需要下界衰减）：chunkwise 并行时，遗忘率在一个 16-token 的 tile 内要连乘 16 次。若每步 log-decay 可以到 $-\infty$（Kimi Linear 的无下界负 Softplus），16 步累积可能到 e^{-200}——数值下溢，迫使对角 tile 退化为逐位置对的标量计算，Tensor Core（专门做矩阵乘的硬件单元）用不上。K3 给 log-decay 加下界：$g_t^h = g_{min} \cdot \mathrm{Sigmoid}(e^{A_h} z_t^h)$，$g_{min} = -5$，于是 16 步累积 log-decay $\in (-80, 0)$，倒数缩放因子 $< e^{80}$，仍在 BF16 动态范围内——**对角 tile 也能整块矩阵乘**。一个数值技巧换来硬件利用率质变。



(a) Log-decay parameterization.

(b) Diagonal-tile computation.

图：KDA 下界衰减与 chunkwise 计算

怎么看这张图：(a) decay 参数化曲线：Kimi Linear 无下界负 Softplus vs K3 带下界缩放 Sigmoid（红，$g_{min} = -5$），后者永不贴零；(b) 对角 tile 从逐位置对标量计算变成整块矩阵乘。

【论文实战（Gated MLA 与 NoPE）】 K3 的 24 层全局注意力仍用 MLA（缓存压缩 latent），但**全部用 NoPE（无位置编码）**——位置/新近性敏感性由 KDA 层提供，MLA 只管全局内容交互；好处是扩上下文时不用再调 RoPE base 或 YaRN 插值。训练时注意力输出保持 FP32 以修正 flash-attention 的有偏舍入误差。

【论文实战】 代价记住两条：长期记忆会被磨掉，所以配周期性全局注意力；以及 chunkwise 并行的数值陷阱（见上）。**再算给你看（cache 对比）**：1M token 上下文，K3 的 69 个 KDA 层每层只存固定状态 $S \in \mathbb{R}^{d_k \times d_v}$（与上下文长度无关，约数十 KB 级），24 个 Gated MLA 层存压缩 latent。对比 5.1 节 MHA 同长度下每 token 480KB × 100 万 ≈ 480GB 的恐怖账本，混合路线把"随上下文增长的存储"压缩了三个数量级以上——**1M 上下文从"数据中心级难题"变成"单机可服务"**，这就是 KDA 路线的全部意义。

5.6 谱系总结

变体	KV cache/token/层	核心思路	省了什么	代价	使用者
MHA	$2n_h d_h$（V3 等效 32768 元素）	每头独立 K/V	—	cache 巨大	经典 Transformer
MQA	$2d_h$	全头共享 K/V	cache $\div n_h$	质量损失明显	早期推理优化
GQA	$2gd_h$	分组共享	cache $\div (n_h/g)$	轻微损失	Llama 等
MLA	512+64=576 元素	低秩联合压缩 + 解耦 RoPE	cache ÷~57	训练仍 MHA 形态；Muon 下需 Muon Split	V3/R1/K2/K2.5、GLM-5(MLA-256)
DSA	MLA latent + top-k	稀疏选择	长上下文计算 1.5–2×	需训练-推理一致性设计	V3.2、GLM-5
CSA/HCA	条目数 ÷4 或 ÷128 + top-k	序列维压缩 + 稀疏	1M 下 FLOPs 3.7×↓、cache 9.5×↓	局部细节靠滑窗补	V4

CSA2	三者再 × 跨层复用	层维复用	cache 再 ÷4（890 B/token）	层间依赖设计复杂	V4.1
KDA	0（固定循环状态）	线性注意力 delta rule	cache 归零	长期记忆会遗忘；需配周期全局注意力	K3（3:1 混合 Gated MLA）

> 【小白 **FAQ**】
>
> - **Q：MLA 是不是就是 GQA 的变体？** 不是。GQA 是"几个头共享同一份 K/V"（离散共享、有信息损失）；MLA 是"所有头从一个低维 latent 各自解压出 K/V"（连续压缩、理论上无损）。两者在"每 token 存的字节数"上可以调到相同，但 MLA 保留了每头独立的 K/V。
> - **Q：稀疏注意力会不会漏看重要内容？** 这就是 indexer 存在的原因：它专门学"什么重要"。GLM-5 的 SFT loss 与 dense 完全重合，说明 top-k 选择抓住了关键条目；召回率实测 99.7%（V4 的 index 分数从 FP32 量化到 BF16 后）。
> - **Q：线性注意力是不是要取代 Transformer？** 短期内是混合而非取代：K3 自己也是 3 KDA + 1 MLA 的混合。线性注意力赢在长上下文成本，全局 softmax 注意力赢在精确检索，混合各取所长。

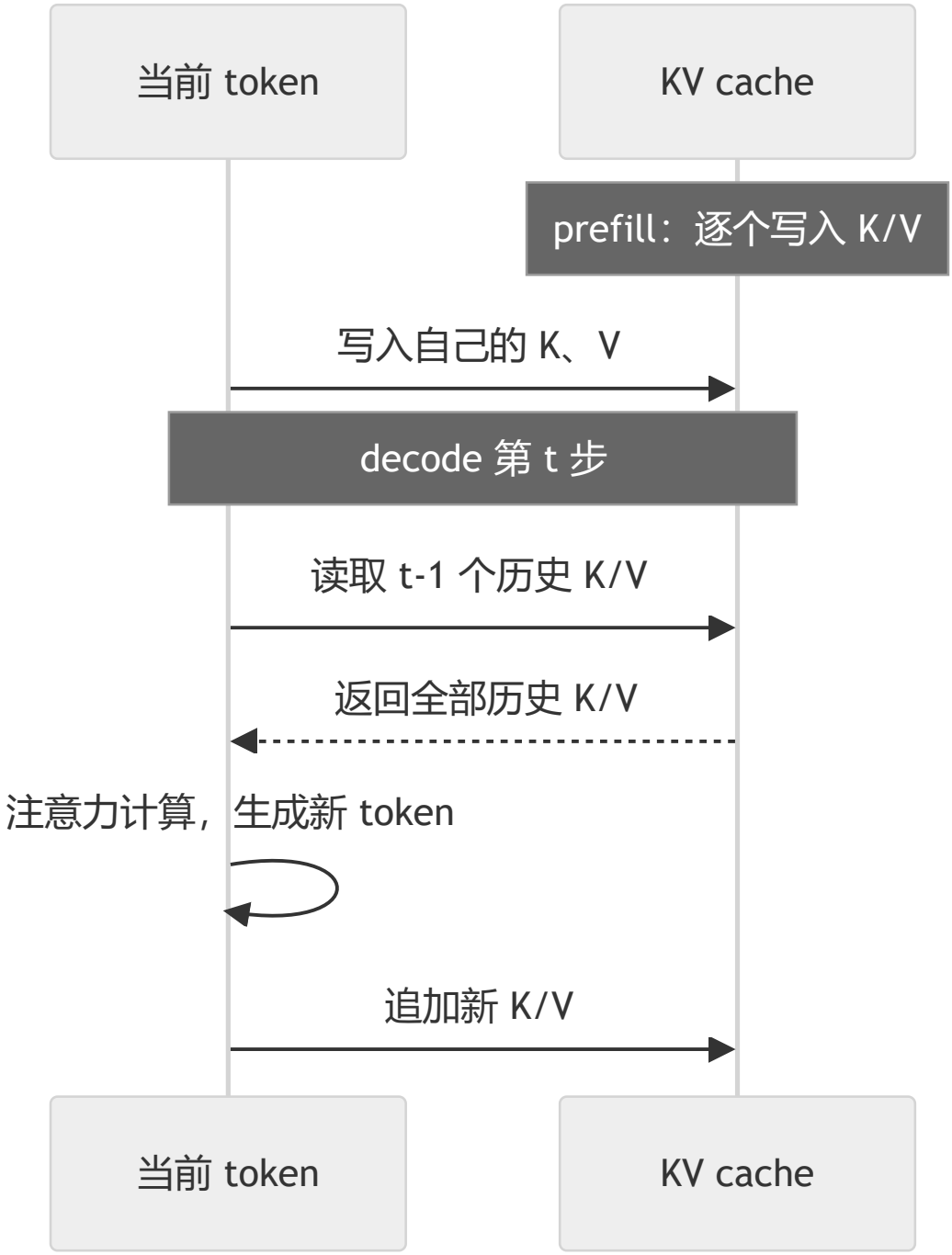
---

6. KV cache 详解：推理成本的头号账本

> 第 5 章一路都在"砍 KV cache"，本章把这本账彻底算清：它为什么存在、到底多大、各家分别砍了哪个因子、最后怎么量化到 4 bit。

6.1 它为什么存在

【大白话】 考试时做阅读理解，允许带一张小抄：把每段的要点抄上去，后面每写一句话要回读前文时，只看小抄不重读原文。KV cache 就是这张小抄——**用显存换时间**。但小抄纸就那么大（显存），文章越长越抄不下；而且 decode 每写一个字都要把小抄整张读一遍（带宽瓶颈，第 2.3 节）。反过来如果不用 cache：每写一个新词都把前文从头重算一遍，生成 N 个词的总计算量从 $O(N^2)$ 涨到 $O(N^3)$ 级别——没有任何服务承受得起。所以 KV cache 不是优化，是"没有它就没有现代大模型服务"的基础设施。



6.2 大小怎么算

【慢慢推导】 经典 MHA 的 KV cache 总量：

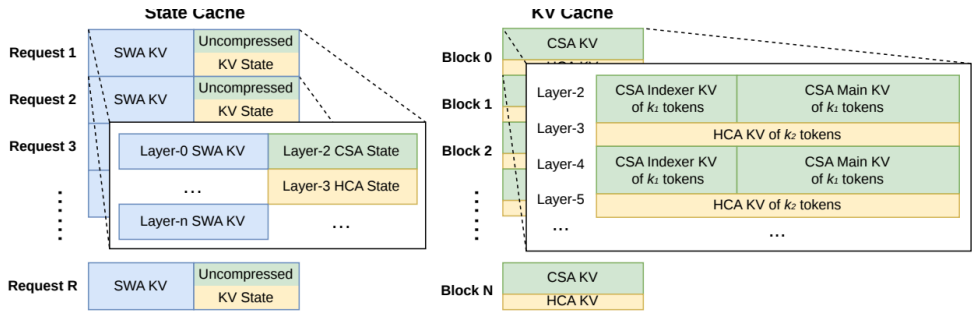
$$\text{Bytes} = 2 \times L \times N \times n_h \times d_h \times b$$

L=层数、N=上下文长度、b=每元素字节数。**算给你看**：61 层、128 头、128 维、BF16（V3 等效 MHA 形态）：每 token $2 \times 61 \times 128 \times 128 \times 2 \approx 4$ MB；100 万 token ≈ **4 TB**——任何单卡都装不下。这个公式的五个因子被各家轮流砍了个遍：

- 砍 n_h：MQA/GQA（共享 K/V）；
- **砍每条条目大小**：MLA 压到 576 元素 → 61 层每 token 约 70 KB（BF16）；
- **砍条目数**：CSA 每 4 个 token 合并 1 条、HCA 每 128 个合并 1 条、DSA/CSA 再 top-k 稀疏；
- **砍层数** L：CSA2 跨层复用（6 层一组只有 1 层真算）；K3 直接让 69/93 层没有 KV cache；
- 砍 b：FP8、FP4 存储。

6.3 各模型的实战成绩

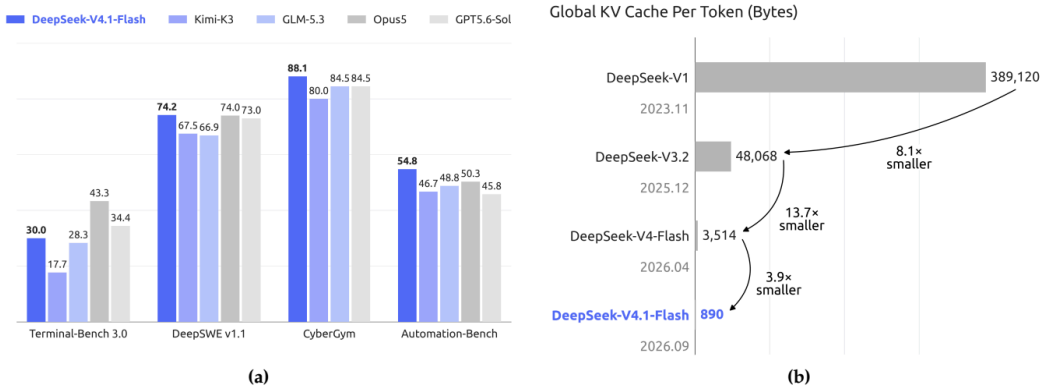
DeepSeek-V4：1M 上下文下 KV cache 仅为 V3.2 的 **1/9.5（Pro）/ 1/13.7（Flash）**，相对 BF16 GQA-8 基线约为 **2%**；RoPE 维度保 BF16、其余 FP8 再省近半。



图：DeepSeek-V4 的 KV cache 布局

怎么看这张图：cache 组织成两部分——经典 KV cache（CSA 的 indexer/main KV 与 HCA KV，按 Block 分页，类似操作系统虚拟内存页表，定长块便于分配回收）和 state cache（滑窗 SWA 的 KV + 尚未压缩的 token，每请求一个固定大小块）。

DeepSeek-V4.1 的 890 B/token 是三因子相乘的总账：层维压缩（CSA2 跨层复用）× 序列维压缩 × 位宽（FP4）= 全局 KV cache **890 字节/token**（HBM 常驻），是 V4-Flash 的 1/4、V1 的 **1/437**；配合 SWA Bounded Replay（滑窗 KV 不持久化、缺了只重放最后 128 个 token 重建近似状态），持久化 cache（SSD/host）再降到 V4-Flash 的 1/8。



图：历代 DeepSeek 模型每 token 全局 KV cache 对比

怎么看这张图：(b) 横向条形是历代每 token 全局 KV cache 字节数，注意是对数尺度——每一档都是数量级下降，V4.1-Flash 的 890 B 比 V1 小 437 倍；(a) 是 agentic benchmark 性能，说明压缩没有以能力为代价。

Kimi K3 的不同路线：69 层 KDA 无 KV cache（固定循环状态 S，还便于 state-aware prefix caching 跨请求复用），仅 24+1 层 Gated MLA 有压缩 cache——1M 上下文下 cache 极小。

6.4 量化 KV cache：最后一个乘数

【大白话】 搬家时用越来越小的箱子装书：先少带书（稀疏 top-k），再把百科全书换成口袋本（序列压缩、层复用），最后把口袋本影印成缩微胶片（FP4）——但地图和指南针（RoPE 维度、SWA 局部 KV）必须保持清晰，缩微了就找不到路。

【慢慢推导】 **算给你看（890 B/token 怎么乘出来）**：V4.1-Flash 的全局 KV 每条 latent 512+64 维。三层因子相乘——层维：decoder 侧 5 组×4 层仅 1 层产出真 KV，等效层数 ÷4；序列维：encoder m=2 压缩；位宽：FP4（0.5 B）替代 BF16（2 B）÷4。从 V3 系 MLA 的约 70KB/token（576 元素 × 61 层 × 2B）出发，逐因子砍：砍位宽 → 约 17.5KB（FP8/FP4 混合）；砍序列维与条目稀疏 → 约 3.6KB（V4-Flash 水平）；砍层维复用 → **890 B**。每一刀对应前面章节的一项技术，这张"瀑布账"是全文的浓缩。

【论文实战】 精度梯队的排布逻辑是"容忍度决定位宽"：

- **V4**：latent 主体 FP8，RoPE 的 64 维保 BF16——旋转位置编码对精度最敏感；indexer 打分路径用 FP4——打分只用于排序（挑 top-k），对误差天然容忍，实测 index 分数 FP32→BF16 后 top-k 选择器提速 2 倍、KV 召回率保持 **99.7%**。
- **V4.1**：FP4 推到 main KV（E2M1 + 每 16 通道 E4M3 scale，去掉二级全局 scale，论证见 4.3 节）；反直觉的细节是"**RoPE 之后再量化**"——先量化再旋转的收益微乎其微，反而增加 decode 开销。SWA 的 KV 只覆盖最近 128 个 token、对局部细节最敏感，保留 FP8。

> **【小白 FAQ】**
>
> - **Q：KV cache 和显存里的参数是一回事吗？** 不是。参数是模型的"知识"，所有请求共享、固定不变；KV cache 是每个请求的"对话历史笔记"，随上下文增长、请求间不共享（除非 prefix caching）。
> - **Q：cache 压得这么小，模型会不会"忘事"？** 压缩的是存储形式不是内容（MLA 无损解压）；稀疏丢的是被 indexer 判定为不重要的条目；KDA 才是真的"会遗忘"——所以配了周期性全局注意力兜底。

---

7. MoE：混合专家，用"稀疏激活"换参数规模

> 第 2 章的账本显示：模型能力跟参数量正相关，但成本跟"激活参数量"正相关。有没有办法让两者脱钩？**MoE（Mixture of Experts，混合专家）**就是答案，也是七篇论文全部采用的架构。

7.1 直觉与路由

【大白话】 与其养一个什么都会的通才（dense FFN），不如开一家 256 个科室的大医院：导诊台（Router）给每个病人（token）打分，挑出最对口的 8 个科室（Top-K 路由），病人只去这几科看病，最后把几份诊断按权重汇总。全院"知识总量"（总参数）可以做得巨大，单个病人的花销（激活参数）只看 8 个科室。

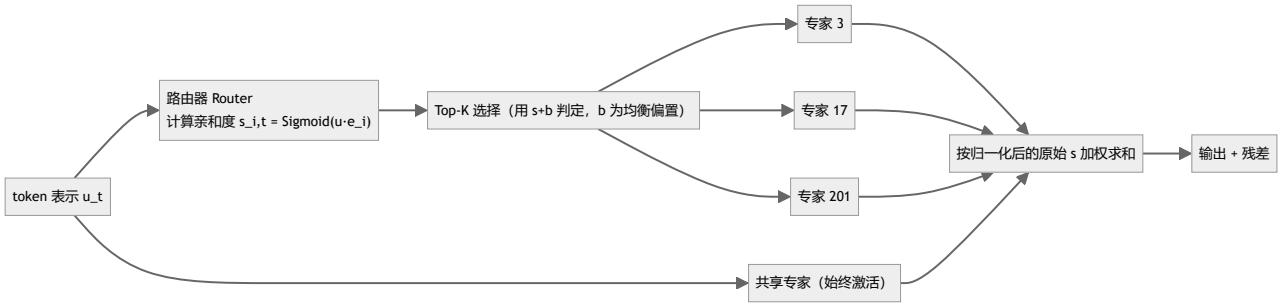
【细粒度专家为什么更好？】 同样总参数量，"4 个大专家选 1 个"和"256 个小专家选 8 个"的组合自由度天差地别：后者有 $\binom{256}{8} \approx 4 \times 10^{14}$ 种专家组合，相当于给每类 token 定制一个"专科会诊团"；前者只有 4 种选择。组合数越大，模型对知识的"切分-重组"越精细——这是 DeepSeekMoE 把专家做小（中间维仅 2048）的理论动机，K3 更是推到 896 专家选 16（组合数天文数字），代价就是第 7.3 节那套通信与稳定性工程。

【慢慢推导】 **算给你看（MoE 的稀疏账）**：V3 每层 1 个共享专家 + 256 个路由专家，专家中间维 2048。一个路由专家的参数量约为 $3 \times 7168 \times 2048 \approx 44\text{M}$（SwiGLU 三个矩阵），256 个专家 ≈ 11.3B；加上共享专家，单层 MoE 约 11.3B 参数。若换成等宽 dense FFN 只有约 88M 参数。每个 token 只激活共享专家 + 8 个路由专家 ≈ 396M 参数——**参数扩大 128 倍，计算只扩大 4.5 倍**。全模型 58 层 MoE 累积出 671B 总参 / 37B 激活的规格，就是这么来的。

【正式定义】 DeepSeekMoE 的两项改进：**细粒度专家**（每专家中间维仅 2048，是传统专家的 1/4 左右——切得更碎、组合更灵活，如同把"大内科"拆成"呼吸/消化/心内/肾内"）和**共享专家**（1 个始终激活，负责"的、是、了"这类通用知识，避免每个路由专家重复学常识）。路由打分用 sigmoid 亲和度：

$$s_{i,t} = \text{Sigmoid}(u_t^T e_i)$$

e_i 是专家 i 的质心向量。V2 用 softmax，V3 改 sigmoid 并在选中专家间归一化；V4 又改成 $\sqrt{\text{Softplus}(\cdot)}$。

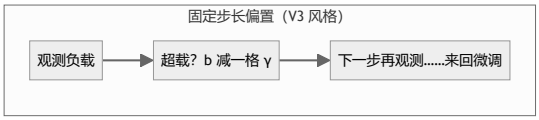


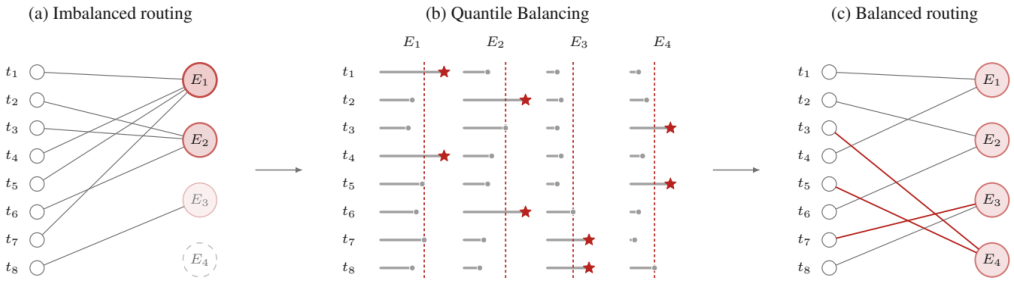
7.2 负载均衡：三代方案

【大白话】 热门科室排长队、冷门科室医生闲到"饿死"（参数得不到训练）。**先算给你看问题有多严重**：256 个专家、每 token 选 8 个、一个 batch 100 万 token——理想均衡是每个专家收 $10^6 \times 8/256 = 31,250$ 个 token。但实测无干预时热门专家可能收 10 万以上而冷门专家为 0：热门专家所在的 GPU 算冒烟、其他卡空等（木桶效应，整步时间由最忙的卡决定），冷门专家的参数还永远得不到梯度。三代管理方案：

1. **辅助损失（auxiliary loss）**：给训练目标加一项惩罚负载不均的损失。问题：惩罚项干扰主任务——像为了各科室接诊量平均，强迫导诊台把胃病病人分去骨科，病看不好（掉点）。
2. **Aux-loss-free 偏置（V3 首创）**：每个专家一个只用于 Top-K 判定的偏置 b_i，门控加权值仍用原始 $s_{i,t}$——**排班倾向悄悄调，看病本身不受污染**。每步监控负载：超载专家 $b_i \leftarrow b_i - \gamma$、欠载 $+\gamma$（$\gamma = 0.001$，前 14.3T token，之后置 0），另加极小的 sequence-wise 平衡损失（$\alpha = 0.0001$）防单序列内极端失衡。V4.1 扩展出**模态专属偏置**：图像/文本 token 各维护一套偏置、按各自模态负载独立更新。
3. **Quantile Balancing（K3）**：固定步长像"每次只调一格温度"，要么调不到位、要么来回震荡。QB 一步到位：取 Top-$(k+1)$ 路由的第 $k+1$ 名为截断线 α_i，把专家偏置直接设为边际分数 $s_{i,j} - \alpha_i$ 的 $(1 - k/n)$ 分位数（再去均值）——**单次前向即精确匹配目标负载**；全局 batch 上用直方图 all-reduce 估计分位数，每专家只通信几百个 bin。

【慢慢推导】 **算给你看（QB 一步到位的含义）**：8 个 token、4 个专家、每 token 选 1 个（$k = 1, n = 4$，目标每专家负载 $q = mk/n = 2$）。某专家对 8 个 token 的边际分数排序后为 $(0.9, 0.7, 0.5, 0.3, 0.1, -0.1, -0.3, -0.5)$；要恰好入选 2 次，偏置应让第 2 名压线、第 3 名出局——即偏置 = $-(1 - k/n) = -0.75$ 分位数的相反调整，取分数序列的 0.75 分位（≈0.6）的相反数附近值。一次前向、一次分位数估计，偏置直接落在精确值上；而固定步长法要从当前偏置出发一格一格挪（每步 ±0.001），震荡若干步才收敛，还依赖全局负载的及时反馈。这就是"分位数一步定偏置 vs 符号更新慢慢来"的本质差别。





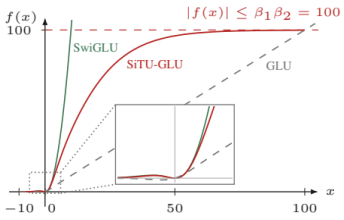
图：Quantile Balancing 负载均衡示意

怎么看这张图：三栏对比，设定 m=8 个 token、n=4 个路由专家、k=1。(a) 原始 Top-k：负载 (4,3,1,0)，深色专家"过热"、虚线专家"濒死"；(b) 分位数计算：找到让负载恰好均分的偏置；(c) 均衡后每专家恰好 2 个 token。对比固定步长的来回震荡，QB 一步到位。

7.3 EP 通信与前沿工程

【论文实战】

- **Node-limited routing（V3）**：每 token 最多发往 M=4 个节点（按节点上最高亲和度之和选节点），按住跨节点通信总量；V4 移除该限制，靠重设计并行策略弥补。V3 全程 **No Token-Dropping**（不丢 token）。
- **Hash routing（V4）**：前 3 层按 token ID 的哈希函数定专家——前几层语义未成形，学习路由不划算，哈希直接均分。
- **LatentMoE（K3）**：896 专家 top-16 太宽，让路由专家在 $\ell = 3584$（$0.5 \times d$）的紧凑潜在空间工作：$z = W_{\downarrow}x$ 降维分发、聚合后 $W_{\uparrow}\mathrm{RMSNorm}(u)$ 投回 d——通信与权重流量直接减半。配套三招稳定 56 倍极端稀疏：聚合后插 RMSNorm 抑制激活爆炸；**SiTU-GLU** 双 softcap 激活（$\beta_1 \tanh(W_g x/\beta_1) \cdot \sigma(W_g x) \odot \beta_2 \tanh(W_u x/\beta_2)$，$\beta_1 = 4, \beta_2 = 25$，近原点近似 SwiGLU、大值有界 $|f| \le 100$）；Quantile Balancing。
- **Engram（V4.1）**：并非所有知识都要参与矩阵乘——196B 参数的 N-gram 哈希条件记忆放 host 内存，靠确定性寻址 + 后台 RDMA 预取在使用前搬到显存（第一模块预取与第一个 Transformer block 计算重叠）。"记忆与计算解耦"：MoE 解决计算稀疏，Engram 解决存储稀疏。

	Gate branch	Up branch	Curve
GLU [26]	$\sigma(x)$	x	
SwiGLU [108]	$x \cdot \sigma(x)$	x	
SiTU-GLU	$\beta_1 \tanh\left(\frac{x}{\beta_1}\right) \cdot \sigma(x)$	$\beta_2 \tanh\left(\frac{x}{\beta_2}\right)$	

图：GLU、SwiGLU 与 SiTU-GLU 曲线对比

怎么看这张图：三条标量响应曲线。SwiGLU 两支都无界，大输入产出 outlier 激活（训练炸点来源）；SiTU-GLU（红）在原点附近紧贴 SwiGLU，大值区域被"软钳制"到有界区间。小图放大原点区域。

> **【小白 FAQ】**
>
> - **Q：专家是不是"一个懂数学、一个懂编程"这样分工的？** 不完全是。分工是自发涌现的，部分专家确实表现出领域倾向，但更多是按句法/语义模式分工，不可直接解读。
> - **Q：激活 8 个专家，那剩下的 248 个这步是不是白训练了？** 不是。一个 batch 有几十万 token，每个 token 选不同的 8 个，合起来所有专家都被频繁训练。负载均衡保证没有专家被长期冷落。

---

8. 训练算法：从下一词预测到强化学习

> 前面七章都在讲"身体"（架构与系统）；本章讲"灵魂怎么注入"——预训练目标、优化器、以及让模型学会推理的强化学习。

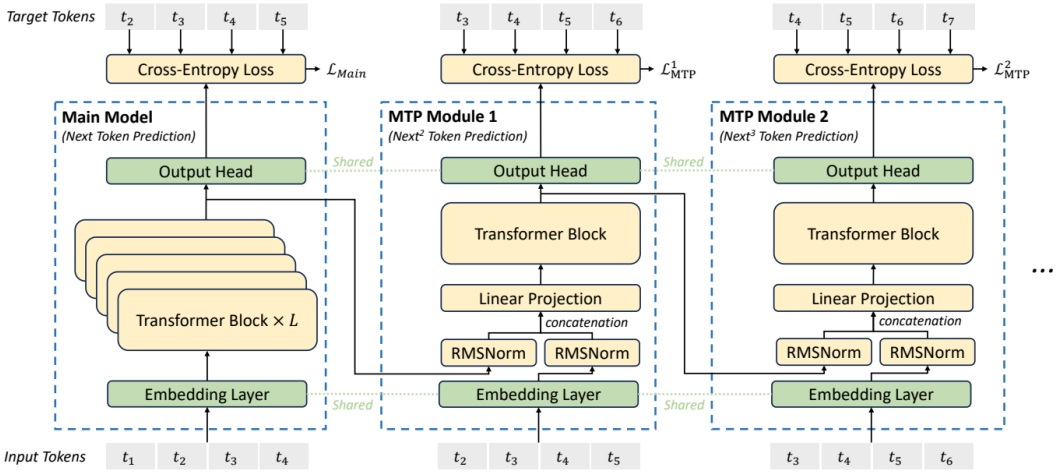
8.1 预训练目标与 MTP

【大白话】 预训练就是玩"接龙"：给前半句猜下一个词，猜错就修正（交叉熵损失）。**MTP（Multi-Token Prediction，多 token 预测）**把规则改成"猜下两个词"——只猜下一个像"走一步看一步"，多猜一步像下棋"想两步"，逼迫模型学更长程的规划。**算给你看（为什么 MTP 几乎免费）**：V3 的 MTP 只有 1 个额外 Transformer block + 一个投影矩阵，参数量相对 61 层主干约 $1/61 \approx 1.6\%$；而 $\lambda = 0.3$ 的辅助损失权重意味着它只贡献约三成的额外监督信号——用百分之一点几的参数换"想两步"的训练信号，性价比极高。V4.1 放弃 MTP 转用 DSpark 则是另一笔账：MTP 的训练收益递减，而独立训练的投机模块（backbone 冻结、梯度不回传）可以把全部容量用在"猜得准"上，还能加 confidence head 做系统级调度。

【正式定义】 NTP 目标 $\mathcal{L} = -\sum_t \log P_\theta(t_{i+1}|t_{\le i})$。V3 的 MTP（深度 D=1）：一个额外模块，含共享 embedding、共享输出头、1 个 Transformer block 和投影矩阵 $M_k \in \mathbb{R}^{d\times 2d}$：

$h_i'^k = M_k[\text{RMSNorm}(h_i^{k-1}); \text{RMSNorm}(\text{Emb}(t_{i+k}))]$

保持完整因果链；损失 $\mathcal{L}_{MTP} = \frac{\lambda}{D}\sum_k \mathcal{L}_{MTP}^k$，$\lambda$ 前 10T token 为 0.3、之后 0.1。**【推理时 MTP 模块可丢弃，或复用做投机解码（speculative decoding）】**：小模块先连续猜几个词、大模型并行验证，接受的词等于"白赚"。**算给你看**：普通 decode 生成 5 个词要 5 次完整前向（每次读全部参数）；投机解码先由小模块一次猜 5 个草稿词，大模型**一次前向并行验证 5 个位置**——若平均接受 2.76 个词（GLM-5 实测 accept length 2.76，DeepSeek-V3.2 为 2.55），则每次大模型前向净产出 2.76 词而非 1 词，decode 吞吐近乎 ×2.76。由于 decode 是带宽瓶颈（2.3 节），"一次参数读取换多个词"正中要害——batch 越小（如 RL rollout），收益越大。



图：DeepSeek-V3 的 MTP 实现

怎么看这张图：主干输出表示 h_i，MTP 模块把 h_i 与下一个 token 的 embedding 拼接、过投影和独立 Transformer block 后预测第 $i+2$ 个 token。注意箭头是链式的——每个深度的预测保持完整因果链，而非多头并行乱猜。

8.2 优化器：从 Adam 到 Muon 家族

【大白话】 优化器是"教练的纠偏方式"。**Adam** 像细心教练：记录每个学员（参数）过去每次失误的幅度和波动，个性化地决定这次纠正多大劲——代价是给每人建一份详细档案（一阶+二阶动量，显存翻倍）。**Muon** 换了哲学：不管过去失误如何，每锤的力道都标准化（把更新矩阵的奇异值都正交化推到 1）——更新更均匀，token 效率更高。

【慢慢推导】 **Muon 正交化的直觉**：一个更新矩阵 G 可能"有的方向步子大、有的方向步子小"（奇异值差异大）。理想更新应该各方向均匀，即让 G 变成半正交矩阵。精确做法 SVD 太贵，用 **Newton-Schulz 迭代**近似：反复做 $X \leftarrow aX + b(XX^T)X + c(XX^T)^2X$ 这样的矩阵多项式（只需要矩阵乘法，GPU 友好），几步就把奇异值推向 1。**算给你看（V4 的 hybrid NS）**：

前 8 步用系数 (3.4445, −4.7750, 2.0315) 激进收敛，后 2 步换 (2, −1.5, 0.5) 精确稳定——"先猛跑后微调"，共 10 步迭代。更新再做 RMS rescale 到 0.18，就能直接复用 AdamW 的学习率，迁移成本极低。配置：动量 0.95、wd 0.1、Nesterov trick；embedding/输出头/RMSNorm 等仍用 AdamW（β=0.9/0.95, wd=0.1）。

【论文实战（Muon 家族分化）】

- **MuonClip + QK-Clip（K2/K2.5）**：钳制注意力 logits，稳住 Muon 训练。
- **Muon Split（GLM-5）**：上投影矩阵按头拆分分别正交化，救回 MLA（见 5.3 节）。
- **Head-wise / Per-Head Muon（V4.1 / K3）**：Q/K（V4.1）或 Q/K/V（K3）按头逐头正交化，避免大梯度头主导共享更新方向；per-head 高瘦矩阵的 NS 迭代还更便宜。
- **Sinkhorn-balanced 更新（V4.1，用于 embedding/Engram 表/预测头）**：$\Delta_t = \sqrt{n}\, D_r \hat{G}_t D_c$，使更新矩阵行/列 RMS 都约等于 1，近零行掩码；只需动量 buffer 却比 Adam 更好，省一半优化器显存；$\gamma = 0.18$ 匹配 Adam 更新幅度。

【为什么 2026 年 Muon 全面取代 AdamW？】 三个原因：其一 **token 效率**——15T~45T token 的时代，10% 的效率提升就是天文数字的算力；其二 **结构感知**——矩阵参数交给 Muon、向量参数留给 AdamW，好钢用在刀刃上；其三 **可拆性**——正交化粒度本身成了新调优维度。优化器从"通用黑盒"变成"架构协同设计的一部分"。**算给你看（显存对比）**：10B 矩阵参数用 AdamW 需一阶+二阶动量共 $10B \times 8B = 80$ GB 状态；Muon 只需动量一项约 40GB（BF16 则 20GB），且 Newton-Schulz 迭代全是矩阵乘法——正是 GPU 最擅长的操作，10 步迭代的额外算力开销相对整步训练可忽略。省显存、提效率、只用现成算子，Muon 的三样好处凑齐了。

8.3 RLHF、PPO 与 GRPO

【大白话】 预训练让模型"会接龙"，**RLHF（基于人类反馈的强化学习）**让它"答得好"。**PPO** 像请一位随行估价师（critic 价值网络）评估每步棋的好坏——但估价师本人也要训练、会和策略模型一样大，显存与复杂度翻倍。**GRPO** 把估价师辞退：同一道题让模型写 16 份答卷，互相比——比平均分高的就是好答案。

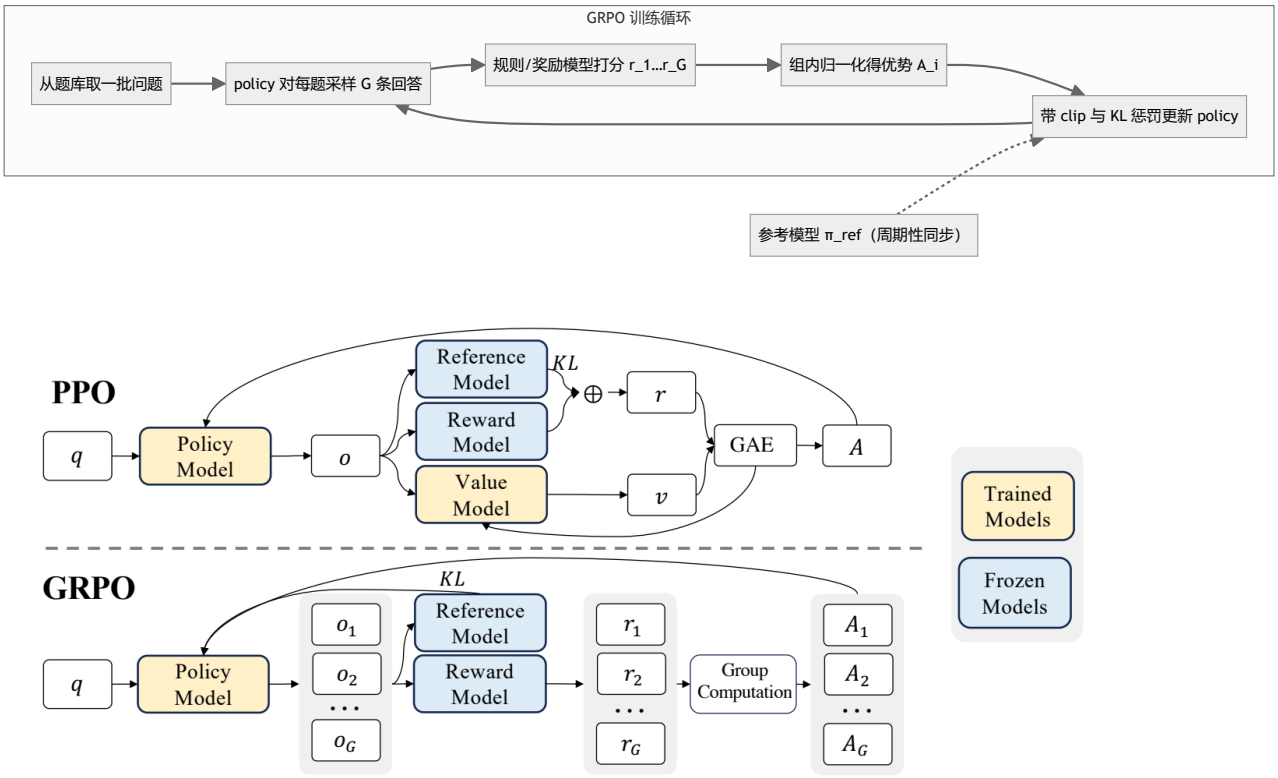
【慢慢推导】 **算给你看（GRPO 优势怎么算）**：某题采 4 条回答，规则奖励打分为 $r = (1, 1, 0, 1)$。组均值 0.75、标准差约 0.433，则优势 $A = (1 - 0.75)/0.433 \approx (0.58, 0.58, -1.73, 0.58)$——得 0 分那条被强烈负向更新，其余温和正向。**注意：全对或全错时优势全为 0**（没有组内差异就学不到东西），这就是 GRPO 偏好"难度适中"题目的原因。

【正式定义】

$$A_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}, \quad \mathcal{J}_{GRPO} = \mathbb{E}\Big[\frac{1}{G}\sum_i \min\big(\rho_i A_i, \text{clip}(\rho_i, 1-\varepsilon, 1+\varepsilon)A_i\big) - \beta D_{KL}(\pi_\theta \| \pi_{ref})\Big]$$

GRPO 与 PPO 逐项差异：critic 网络（有 → 无）；优势估计（GAE 广义优势估计 → 组内归一化）；奖励粒度（逐 token 价值 → 整条回答一个分再归一化）；显存（双模型 → 单模型 + 参考

模型）；KL 估计（PPO 用采样近似 → GRPO 用 $\frac{\pi_{ref}}{\pi_\theta} - \log \frac{\pi_{ref}}{\pi_\theta} - 1$ 的无偏估计）。**算给你看（省多少显存）**：对 671B 的 R1 做 RL，PPO 需要 policy + critic + reference 三个大模型常驻，仅参数（BF16）就约 $671 \times 2 \times 3 \approx 4$ TB；GRPO 去掉 critic，约省 1.3TB——十几张 H800 的显存，还顺带砍掉了 critic 本身的训练开销与"critic 估不准带偏 policy"的误差源。



图：PPO 与 GRPO 对比

怎么看这张图：上栏 PPO 有 value model（critic）参与优势估计；下栏 GRPO 的 critic 消失，换成"Group Computation"——同一 prompt 多条输出的奖励组内归一化直接得优势。少一个大模型，显存和工程复杂度立省。

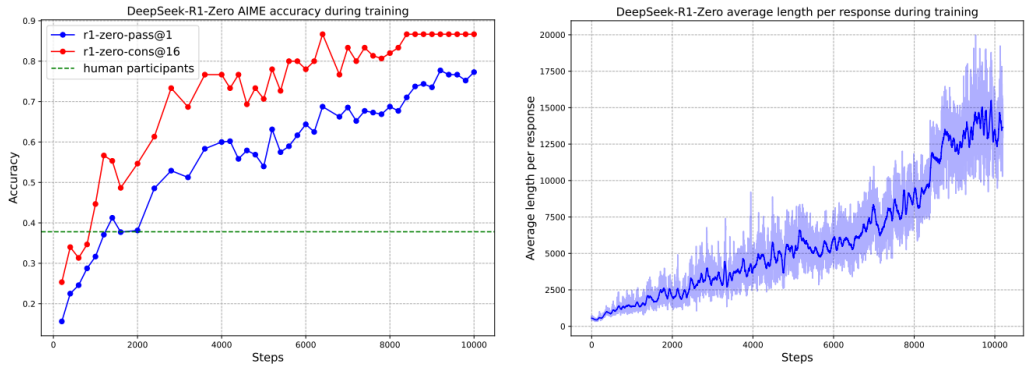
【论文实战（超参细节）】 R1 第一阶段 RL：LR 3e-6、KL 系数 0.001、clip $\varepsilon = 10$、rollout 温度 1.0；每题采 16 条、单条最长 32,768 token；每步 32 题 × 16 = batch 512；每 400 步用最新 policy 替换 reference model；每次 rollout 生成 8,192 条输出、拆 16 个 minibatch 单 inner epoch。

8.4 R1：纯 RL 涌现推理与蒸馏

【大白话】 R1-Zero 干了一件"离经叛道"的事：不先请人教（无 SFT）、不请人示范推理过程（无人类标注），直接给 base 模型出数学题、只按"最终答案对不对"给分。结果模型自己"悟"出了推理——就像只告诉一个学生考试分数、从不讲题，他居然自己学会了打草稿、验算、自我纠错。

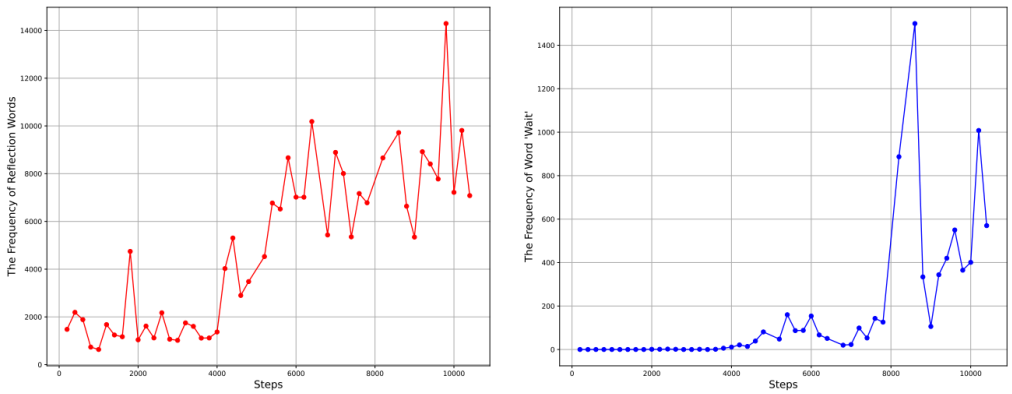
【论文实战】 配方极简：GRPO + 规则奖励（数学用规则验证最终答案如 boxed 格式；代码用编译器跑测试用例；格式奖励要求思考放在 `<think>...</think>`）+ 极简模板（只要求"先 think 再 answer"，不指定解题策略）。**自我进化现象**：AIME 准确率稳步上升；平均响应长度自发增长（learn to think longer，第 8.2k 步附近长度突跳）；出现 **"aha moment"**——自发重新评估、纠错。遗留问题：可读性差、中英混杂、只覆盖推理任务。

【再深一层：为什么是"规则奖励"而不是"更聪明的奖励模型"？】 奖励模型是神经网络，本身是近似——它会给"看起来对"的回答高分，模型很快就会学会"写得看起来对"（reward hacking：答案错但写得自信满满、格式漂亮）。规则奖励（数学验证器、编译器）没有近似空间：对就是对。R1 的经验是**在可验证领域用规则、在不可验证领域（写作、对话）才退而用奖励模型**，且偏好奖励仅在第二阶段 RL 的最后 400 步加入——先把能力练扎实，再谈讨人喜欢，顺序不能反。



图：R1-Zero 的 AIME 准确率与响应长度随训练增长

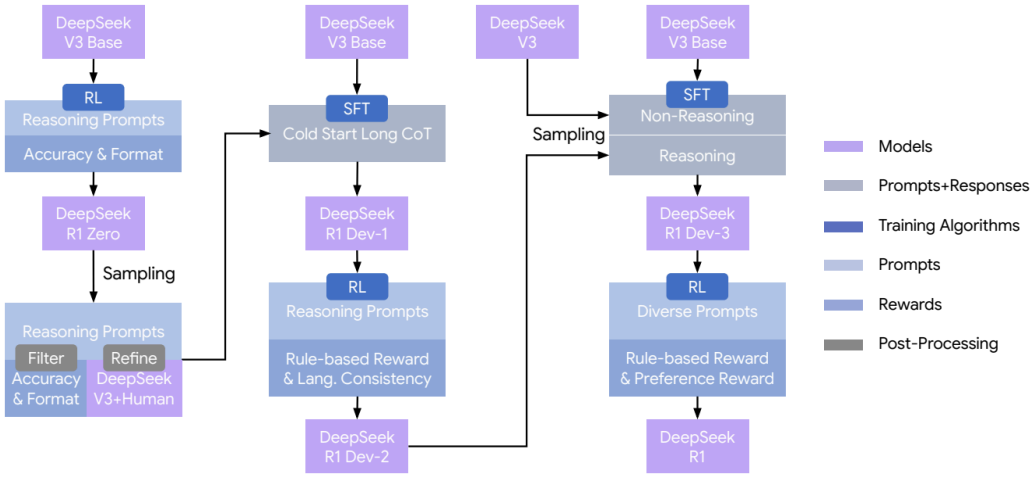
怎么看这张图：(a) AIME 准确率随 RL 步数上升，终点超过人类参赛者平均水平（虚线基线）——全程没人教过它推理；(b) 平均响应长度持续爬升，模型自发学会"想更久"。这两张图是"RL 自主激励推理能力"的标志性证据。



图：训练中推理行为的演化

怎么看这张图：(a) 反思性词汇（re-evaluate、check 等）出现频率随训练上升；(b) 聚焦 "wait" 的出现模式——模型开始在推理中途停下来质疑自己。"aha moment"是被奖励筛选出来的，不是被教出来的。

R1 四阶段流水线（修复 R1-Zero 的可读性）：①冷启动 SFT（数千条"对话式、人类对齐思维过程"的长 CoT）→ ②推理导向 RL（含语言一致性奖励，轻微损性能换可读性）→ ③拒绝采样 + 混合 SFT（RL 收敛后采样生成推理数据、过滤精炼，与写作/QA 等约 80 万条通用数据混合）→ ④全场景 RL（推理数据仍规则奖励；通用数据用奖励模型——helpfulness RM 66,000 偏好对只看最终 summary、safety RM 106,000 条安全标注；温度降到 0.7，共 1,700 步，偏好奖励仅最后 400 步加入防 reward hacking）。



图：DeepSeek-R1 多阶段训练流水线

怎么看这张图：跟随模型与数据两条线从左到右：V3-Base → 冷启动 SFT → 推理 RL → 采样+混合 SFT → 全场景 RL。关注每个节点"解决了上一阶段什么问题"。

蒸馏：用 R1 的 80 万条数据 SFT 开源小模型（Qwen-1.5B/7B/14B/32B、Llama-8B/70B，2~3 epochs，ctx 32k）。关键对照实验：Qwen2.5-32B-Base 直接大规模 RL（>10K 步）只到 QwQ-32B-Preview 水平，**显著不如**蒸馏——小模型"蒸馏 > 直接 RL"；但突破智能上限仍需更强 base + 更大规模 RL。成绩：R1-Distill-Qwen-32B AIME 72.6 / MATH-500 94.3；R1 本体 AIME 79.8、Codeforces 2029 rating。**失败教训**：Process Reward Model 难规模化（细粒度标注难、易 reward hacking）、MCTS 在 token 级搜索空间爆炸——均未采用。

【再算给你看：RL 的 rollout 规模】 R1 每次 rollout 生成 8,192 条输出、每条最长 32,768 token——一次 rollout 最多产出约 $8192 \times 32768 \approx 2.7$ 亿 token 的推理量，相当于一次小规模推理服务的全部流量；拆成 16 个 minibatch 单 inner epoch 训练。这就是为什么 RL 阶段的成本大头其实在**生成**（推理）而非训练（反向传播），也是 GLM-5/K2.5/K3 都不约而同把"rollout 引擎"当成一等公民来优化（FP8 rollout、投机解码、partial rollout、异步调度）的原因：**RL 时代，推理基建就是训练基建**。

8.5 Agentic RL 与 on-policy 蒸馏（2026 新范式）

【大白话】 2025 年 RL 教模型"做难题"；2026 年 RL 教模型"当员工"——会使用工具、拆解任务、多个分身并行干活（agentic）。训练完还要解决新问题：分身练成的各科技能怎么合并回一个人身上？答案是"让全科医生在自己的病人身上模仿所有专科医生的处方"（on-policy 蒸馏）。

【论文实战】

- **PARL（K2.5，Parallel Agent RL）**：训练 Agent Swarm 的 orchestrator，子智能体冻结、其输出视为环境观测，规避 credit assignment 模糊；奖励 $r_{PARL} = \lambda_1 r_{parallel} + \lambda_2 r_{finish} + r_{perf}$——"实例化奖励"防止模型偷懒退化成串行（串行塌缩）。效果：BrowseComp 78.4%（单智能体 60.6%，超 GPT-5.2 Pro）；执行时间降 **3~4.5 倍**。**算给你看（为什么要实例化奖励）**：若只奖励最终结果 r_{perf}，orchestrator 会发现"串行做也能做对、还省去管理分身的麻烦"——收敛到串行塌缩的局部最优。加入 $r_{parallel}$（实例化越多分奖励越多）相当于给"敢于分权"本身发奖金，把模型推出舒适区；r_{finish}（子任务完成率）则防止"为了拿分狂派分身却不干活"。三项奖励分别对抗三种失败模式：不并行、假并行、做错事——这是多目标奖励设计的教科书案例。
- **slime（GLM-5）**：完全异步 RL 基建，训练/推理解耦，中央 Multi-Task Rollout Orchestrator 调度；**TITO（Token-in-Token-out）**网关保持 action 级 token 对应，消除 re-tokenization 失配；**双面重要性采样**对 rollout log-prob 做 token 级双面 clip $[1-\varepsilon_l, 1+\varepsilon_h]$，不追踪历史 checkpoint 即控制 off-policy 偏差；环境含 10K+ 真实 SWE 任务、终端任务、multi-hop 搜索。**为什么异步是个大问题**：同步 RL 里所有 rollout 必须等最慢的一条完成才能开训（木桶效应），而 agent 任务的耗时时长差异巨大（改一个 bug 可能 1 分钟也可能 1 小时）；完全异步让 GPU 永不空等，但代价是 rollout 用的模型可能已经比当前训练中的模型旧了好几版——这就是 off-policy 偏差，双面重要性采样 clip 就是给这个偏差装上"保险杠"：偏差超出区间就截断，且不需要像传统方法那样保存历史 policy 的完整 checkpoint。TITO 解决的则是另一个隐形坑：agent 与工具交互时文本会被反复拼接，如果两边 tokenizer 行为有细微差异，"模型以为自己说的"和"训练时看到的"会对不上——token 进、token 出的网关把这个对应关系钉死。
- **On-Policy Distillation（OPD，V4；K3 的 MOPD；GLM-5 跨阶段蒸馏）**：先分领域/分推理档位训多个专家，再在**学生自己生成的轨迹**上对齐教师分布 $\mathcal{L}_{OPD} = \sum_i w_i D_{KL}(\pi_\theta \| \pi_{E_i})$（reverse KL，全词表 logit 蒸馏而非 token 级 KL 估计——方差低、更稳定）。V4 用它完整替代混合 RL 阶段；GLM-5 的变体把 GRPO 优势项替换为 $\hat{A}_{i,t} = \text{sg}[\log \pi_{teacher}/\pi_{train}]$，group size 直接设为 1。

【小白 FAQ】

- **Q：为什么奖励不用神经网络打分而要用规则？** 规则奖励（编译器、数学验证器）不会被骗；学习型奖励模型会被"讨好"（reward hacking），比如写长篇空话骗高分。R1 只

> 在通用领域用奖励模型，且很晚才加入。
>
> - **Q：RL 会不会把模型练"野"了？** 会——R1-Zero 的中英混杂就是。所以 R1 加语言一致性奖励、用四阶段流水线兜底；K3 对超预算的啰嗦输出直接奖励 −1 防 verbosity hacking。

---

9. 七篇论文逐一精讲

> 前面八章是"按技术专题横切"；本章"按论文纵切"，每篇按同一结构讲：**想解决什么问题（小白版动机）→ 核心创意一句话 → 逐个创新点拆讲（四层递进）→ 在演进中的位置**。完全没读过原文也能跟上。

9.1 DeepSeek-V3（arXiv 2412.19437）：效率架构的"奠基之作"

想解决什么问题：2024 年底的困境是——frontier 级模型被认为需要上万张顶级 GPU 和数亿美元，普通人玩不起。V3 要问：**能不能用受出口管制的"缩水卡"（H800，带宽被阉割）和几百万美元，训出第一梯队模型？**

核心创意一句话：架构（MLA + 细粒度 MoE）、精度（FP8）、系统（DualPipe + 定制并行）、算法（MTP + aux-loss-free 路由）四层协同优化，每一层都省出一个数量级的零头，叠起来就是总成本数量级的下降。

创新点拆讲：

1. **MLA**（四层递进见 5.3 节）：KV cache 压缩约 57 倍（32768 → 576 元素/token/层）而性能不降，让 671B 模型的长上下文推理在工程上可行。
2. **Aux-loss-free 负载均衡**（7.2 节）：1 shared + 256 routed、top-8、sigmoid 打分 + 动态偏置（γ=0.001）+ node-limited routing（最多 4 节点）+ 微弱 sequence-wise 损失（α=0.0001），全程不丢 token。
3. **FP8 混合精度训练**（4.2 节）：首次在 671B 规模验证，tile 1×128 / block 128×128 细粒度缩放 + FP32 定期累加，相对 BF16 损失误差 < 0.25%。
4. **DualPipe + 16PP/64EP/ZeRO-1**（3.6 节）：不用 TP，通信完全隐藏。集群 2048 张 H800（节点内 NVLink 160GB/s、节点间 IB 50GB/s），总训练成本 **2.788M GPU 小时 ≈ 557.6 万美元**。训练稳定性同样惊人：14.8T token 全程无一次不可恢复 loss spike、无 rollback。

演进位置：V3 最大的贡献不是任何单点技术，而是把"效率"提升为与"能力"平起平坐的设计目标，并给出完整工程账本。后文所有论文都在这份账本的科目上添砖加瓦——读懂 V3 就拿到读懂

其余六篇的钥匙。

补充几个常被忽略但极具复制价值的工程数字：学习率采用三段式——2K 步线性 warmup 到 2.2e-4，恒定至 10T token，cosine 衰减到 2.2e-5（4.3T token），最后 500B token 先 2.2e-5 再降到 7.3e-6；batch size 从 3072 渐进升到 15360；梯度裁剪 1.0；参数初始化 std=0.006；优化器 AdamW（β1=0.9, β2=0.95, wd=0.1）。长上下文用两阶段 YaRN 扩展：4K→32K→128K。后训练 SFT + GRPO 之外还有两个巧思：从 **R1 蒸馏长 CoT 推理能力**回 V3（蒸馏同时控制输出风格与长度），以及 **Self-Rewarding**——V3 自己充当生成式奖励模型给自己打分。这套"超参台账"的价值在于：它是 671B 规模上被验证过的稳定配方，后续 V4/K3 的超参都以它为基线微调（如 V4.1 的 LR 2.6e-4、K3 的 cosine 优于 WSD 结论），足见其锚点地位。

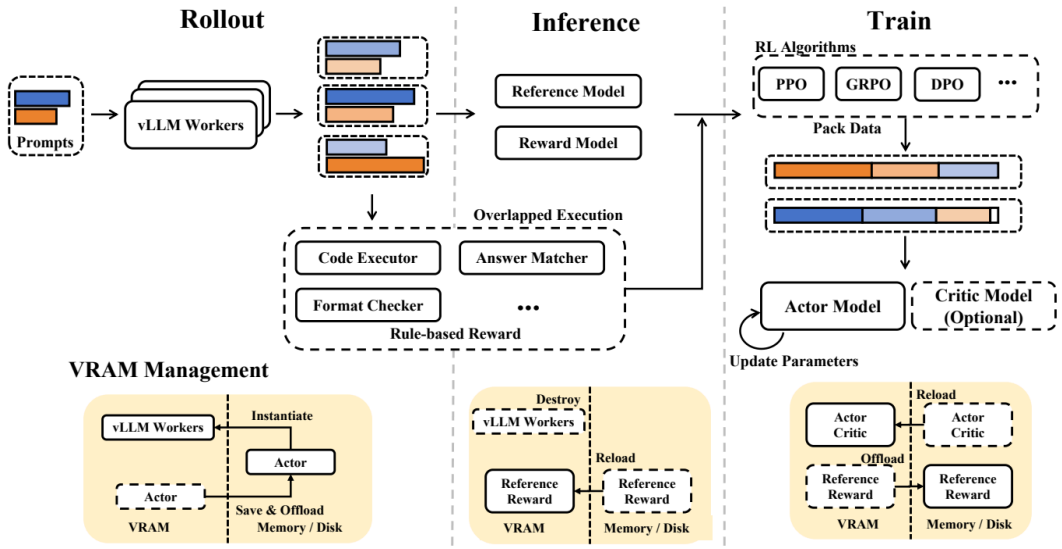
9.2 DeepSeek-R1（arXiv 2501.12948，Nature 版）：把"后训练"变成第一公民

想解决什么问题：V3 解决了"便宜地训出强 base"，但 base 模型只会接龙，不会推理。当时共识是"必须先人工示范推理过程（SFT）再上 RL"。R1 要问：**不教，模型自己能不能悟出推理？**

核心创意一句话：能——纯 RL（GRPO + 规则奖励）直接在 base 上训，推理能力自己涌现；涌现完再用四阶段流水线把"野性"驯化成"可用"。

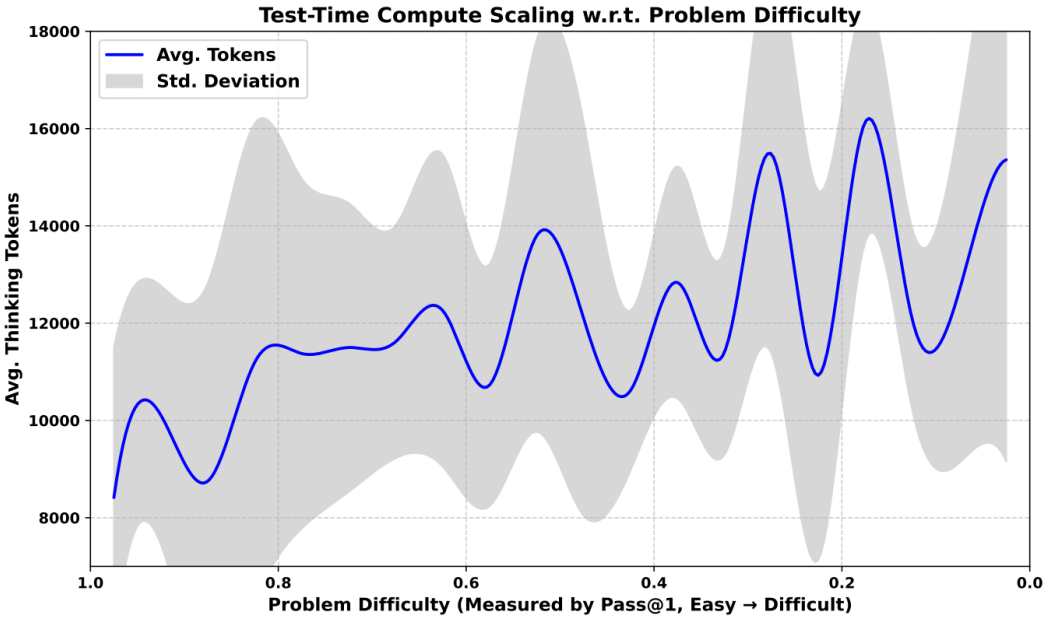
创新点拆讲：

1. **R1-Zero 纯 RL 涌现**（8.4 节四层递进已讲）：GRPO 去 critic 省一半训练显存；规则奖励杜绝 reward hacking；aha moment 与自发变长的思维链是"元认知可以被奖励筛选出来"的直接证据。
2. **四阶段流水线**（8.4 节）：冷启动 SFT 修可读性 → 推理 RL 练能力 → 拒绝采样 + 混合 SFT 扩通用性 → 全场景 RL 做对齐。Dev1/Dev2/Dev3 中间 checkpoint 的设计让每一步的增益可归因。
3. **蒸馏范式**：80 万条数据蒸馏 6 款小模型（1.5B~70B），并用对照实验确立"小模型蒸馏优于直接 RL"——把大模型的探索成果低成本普惠化，R1-Distill-Qwen-32B 至今仍是蒸馏标杆（AIME 72.6）。



图：R1 的 RL 训练框架总览

怎么看这张图：rollout（采样生成）、奖励计算、GRPO 训练三者如何成环；注意参考模型的位置——R1 每 400 步用最新 policy 替换它，让 KL 约束"跟得上进步"。



图：测试时计算扩展：题越难想得越久

怎么看这张图：横轴问题难度（Pass@1 越低越难）、纵轴答对所需思考 token 数，单调上升——模型会按难度分配思考预算。这是后来所有 reasoning effort 控制（K3 的 effort RL、V4.1 的 Minimal 模式）的思想源头。

演进位置：R1 证明后训练可以是能力的主要来源而非美容步骤，直接改写了行业资源分配——此后 GLM-5、V4、K3 的后训练篇幅都接近甚至超过预训练。

R1 留下的另外两条遗产：其一是**评估方式**——评估最大生成长度设为 32,768 token，说明"给够思考预算"本身就是评估的一部分；其二是**对失败的坦诚**——论文 Discussion 明确记录 PRM 与 MCTS 两条路为什么没走通，这在"只报喜"的技术报告文化里相当罕见，也为后来者省下了重复踩坑的成本。另外值得记住的实操细节：蒸馏模型**只做 SFT 不做 RL**（作者明确把 RL 留给社区），说明蒸馏数据本身已把推理模式"固化"得相当充分——这也间接回答了"蒸馏和 RL 的关系"：蒸馏负责迁移模式，RL 负责在新土壤上再进化。

9.3 DeepSeek-V4（arXiv 2604.19348）：为百万 token 而生的架构换代

想解决什么问题：V3 体系在 128K 上下文很成功，但上下文冲到 1M 时，注意力 $O(N^2)$ 计算和 KV cache 双双爆炸（第 5、6 章的账）。**怎么让"读 100 万字"变得像读 10 万字一样便宜？**

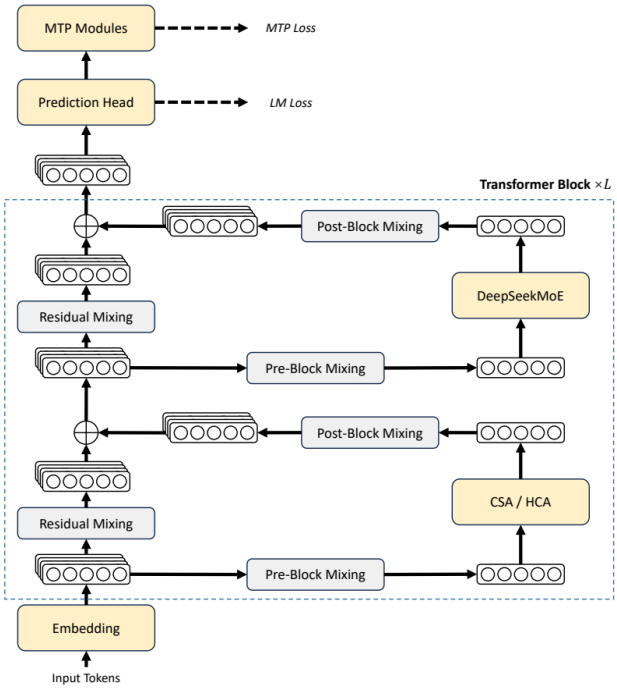
核心创意一句话：放弃纯 MLA，换成"压缩 + 稀疏"的混合注意力（CSA/HCA），配合更宽的残差流（mHC）和 Muon 优化器，做 1.6T 规模的全面换代。

创新点拆讲：

1. **CSA + HCA 混合注意力**（5.4 节）：前 2 层纯滑窗，其余 CSA/HCA 交错。Pro 版 indexer top-k 1024、HCA 压缩率 128。配套精巧细节：Partial RoPE 只加在最后 64 维（KV 兼当 value，输出也用位置 –i 的 RoPE 作用在最后 64 维，使输出携带相对位置信息）；core attention 前对 query 和压缩 KV 做 RMSNorm 防 logits 爆炸（因此 Muon 无需 QK-Clip）；每头一个可学习 attention sink 允许总注意力接近 0。成绩：1M 下推理 FLOPs 仅 V3.2 的 **27%**、KV cache 仅 **10%**；V4-Flash 更达 9.8×/13.7×。
2. **mHC（Manifold-Constrained Hyper-Connections）**：残差流加宽 4 倍（$\mathbb{R}^d \to \mathbb{R}^{4\times d}$），$X_{l+1} = B_l X_l + C_l F_l(A_l X_l)$。风险是加宽残差容易训练发散；约束办法是把混合矩阵 B_l 投影到**双随机矩阵流形（Birkhoff polytope）**上（谱范数 ≤ 1、非扩张、乘法封闭），用 Sinkhorn-Knopp 迭代 20 次（exp 后行列交替归一化）实现。A_l、C_l 用 Sigmoid 保非负有界。**大白话**：残差从单车道扩成 4 车道，但路口红绿灯（B 矩阵）被约束为"只换道、不加油门"，车流不会失控。wall-time 开销仅 6.7%。
3. **Muon 落地 1.6T**（8.2 节），并解决 Muon×ZeRO 矛盾：Muon 要完整梯度矩阵、ZeRO 却切散参数——用 knapsack 算法给每个 rank 分配均衡 bucket（padding < 10%）；MoE 梯度随机舍入量化到 BF16 同步（通信减半），用 all-to-all + 本地 FP32 求和保精度。
4. **训练稳定性工程化**与**多教师 OPD**（8.5 节）：Anticipatory Routing（路由索引用历史参数 $\theta_{t-\Delta t}$ 提前算缓存，打破 backbone 与 routing 同步更新的恶性循环；只在检测到 loss spike 时自动触发，额外开销约 20%）+ SwiGLU Clamping 钳制异常激活。基建配套：TileLang DSL 写融合 kernel、batch-invariant 确定性 kernel 库（训练/推理位级可复现）、DSec 沙箱（单集群数十万并发实例跑 agentic RL）。

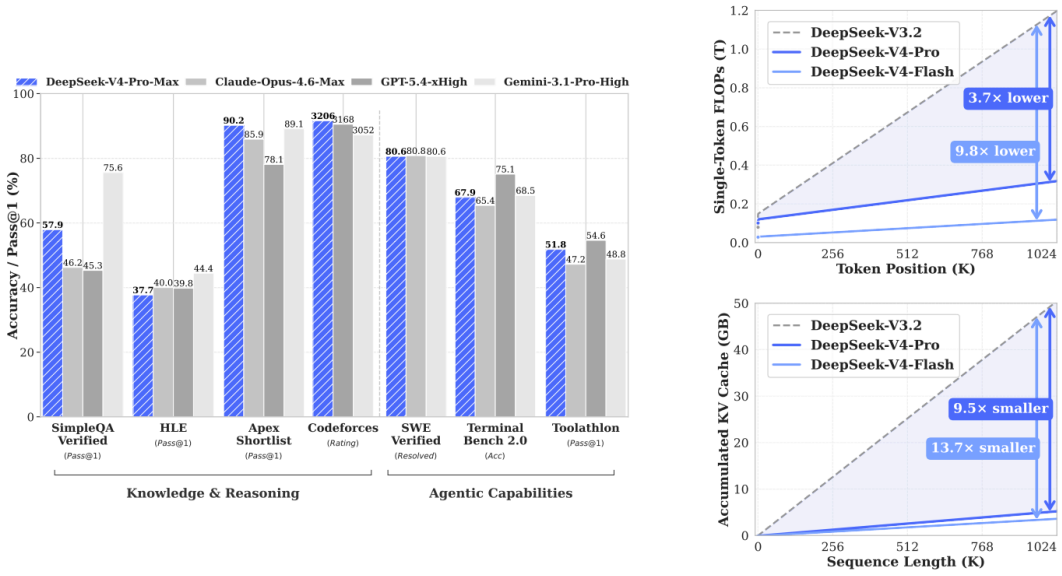
MoE 侧的两个细节也值得细看：亲和度激活函数从 V3 的 Sigmoid 改为 $\sqrt{\text{Softplus}(\cdot)}$（更平滑的正分数，便于配合新的并行策略）；前 3 层用 **Hash routing**（按 token ID 的哈希函数定专家）——V3 前 3 层是 dense FFN，V4 全部换成 MoE，而语义未成形的浅层用学习路由不划算，哈希天

然均分。上下文训练走渐进路线：4K→16K→64K→1M 四段渐进扩展，避免一次性上 1M 的不稳定与浪费。



图：DeepSeek-V4 总体架构

怎么看这张图：自下而上：注意力层 CSA 与 HCA 交错（两种颜色区分）、FFN 全部 DeepSeekMoE、层间残差升级为 Pre/Post-Block Mixing（mHC）、顶部挂 MTP。与 V3 架构图对照看，感受"骨架不变、零件全换"。



图：V4 系列与 V3.2 的推理 FLOPs 与 KV cache 对比

演进位置：V4 把"长上下文"从附加能力变成架构的第一设计目标，并完成了优化器换代（AdamW → Muon）。mHC 与确定性 kernel 这类"看起来不性感"的组件，恰恰是 1.6T 规模能训稳的真正原因。

CP 与确定性的两处工程细节：为压缩注意力定制的 CP 分两阶段通信——rank i 把最后 m 条未压缩 KV 发给 rank $i+1$ 本地压缩（每段只需多处理 $s/m+1$ 条），再 all-gather 收集 + fused select-and-pad 重排，避免了传统 CP 在压缩块边界上的重复计算。**Batch-invariant 与确定性 kernel 库**解决的是科研 reproducibility 的隐形杀手：同一输入两次运行结果不同（浮点归约顺序不确定）。V4 的做法是 token 顺序预处理 + buffer 隔离保证 EP 确定性、mHC 小矩阵 split-k 后确定性归约——训练/推理位级可复现，调试 loss spike 时终于可以确信"差异来自代码而非随机性"。

9.4 DeepSeek-V4.1-Flash：把 KV cache 压到物理极限

想解决什么问题：V4 把 1M 上下文变得"可用"，但还不够"便宜"——KV cache 仍是部署大头；同时模型要走向多模态（看图）和 agent（长任务）。**还能再压一个数量级吗？**

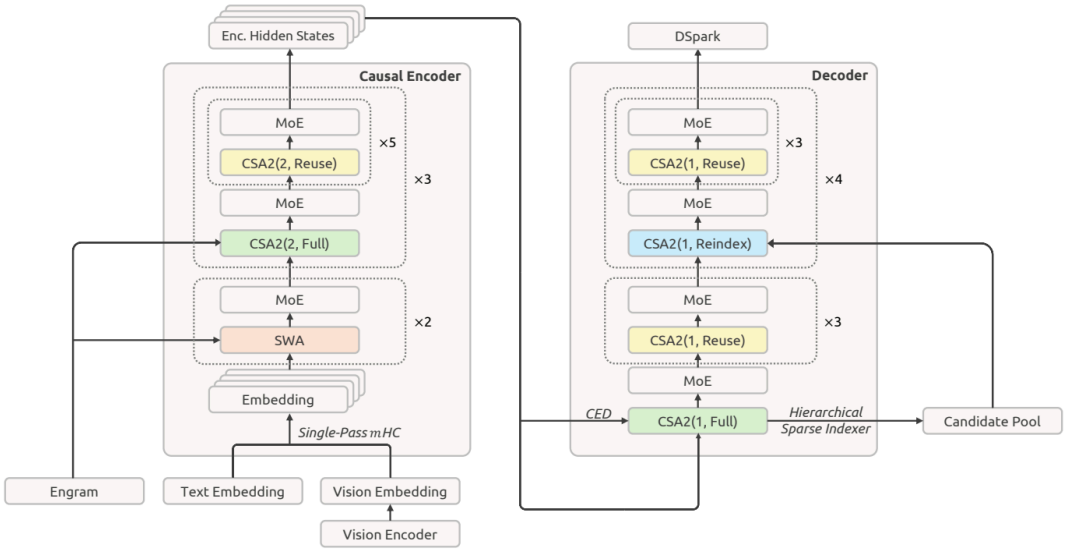
核心创意一句话：把压缩从"条目大小、序列长度"两个维度扩展到第三个维度——**层**（CSA2 跨层复用），再把 prefill 也砍半（CED），配合 FP4 存储，得到 **890 B/token**。

创新点拆讲：

1. **CED（Causal Encoder-Decoder，受 YoCo 启发）**：40 层劈成 20 层 causal encoder + 20 层 decoder。**大白话**：普通模型每层都自己从头算 KV（像 40 个秘书各自把同一份文件重读一遍做笔记）；CED 让前 20 个秘书认真读，后 20 个秘书的"全局笔记"直接从第 20 个秘书的终稿一次投影生成，不再重读。**机制**：decoder 各层全局 KV 由第 L/2 层隐状态 $H_{L/2}$ 经各层专属投影 W_K^l, W_V^l, W_Z^l 直接生成；prefill 只需算前半层，复杂度从 $O(NL)$ 降到约 $O(NL/2)$；SWA 分支仍逐层算（增加局部 KV 的计算深度）。因此 **prefill 每 token 仅激活 8B、decode 激活 16B**。**算给你看**：prefill 一个 1M token 的文档，dense 40 层模型每 token 要过 40 层；CED 只有前 20 层做全量计算，后 20 层的全局 KV 是一次投影的"免费午餐"——若每层激活参数量相近，prefill 的 FLOPs 直接减半，这就是"8B prefill 激活 vs 16B decode 激活"这组不对称数字的来源。
2. **CSA2 三维压缩**：条目大小（GQA/MLA 路线）× 序列维（encoder m=2 / decoder m=1）× **层维（跨层复用）**。三种模式：**Full**（当前层算 main KV + indexer K + Top-K）、**Reindex**（复用最近 Full 层的 KV 和 indexer K，自算 Top-K）、**Reuse**（全复用）。布局：encoder 18 层分 3 组 × 6 层（1 Full + 5 Reuse），decoder 20 层分 5 组 × 4 层（首组 1 Full+3 Reuse，其余 1 Reindex+3 Reuse）。decoder 用**层次化稀疏 indexer**：后续 indexer 只在先前选出的候选池（最多 2048 块 × 8 位置 = 16384 候选）内打分，每 query 打分条目数大降。
3. **FP4 主 KV cache + SWA Bounded Replay**（4.3 / 6.3 节）：HBM 常驻 890 B/token（V4-Flash 的 1/4、V1 的 1/437）；持久化到 SSD/host 再降一半（V4-Flash 的 1/8）。Encoder SWA Bounded Replay：prefix caching 只依赖全局 KV，SWA KV 从持久 cache 移

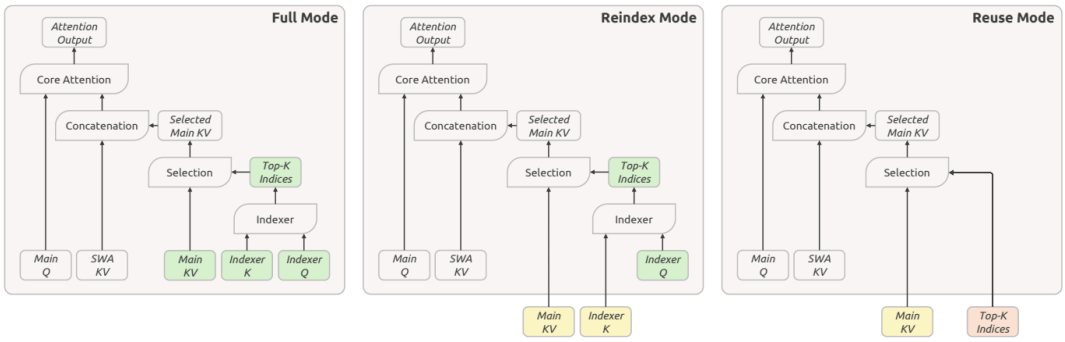
除、缺了只重放最后 128 个 token 重建近似状态；Decoder SWA KV 从不缓存，prefill 计算再近减半——后训练中也模拟同样 replay 做 train-aware 适配。

4. **Engram 与 DSpark**：196B 参数 N-gram 哈希条件记忆（2、3、4 阶、8 个 hash 头、每头索引约 16M 条目的素数长表）放 host 内存 + RDMA 预取（6.3/7.3 节）；MTP 退役，改独立训练的 **DSpark** 投机解码：3 个 Transformer block（滑窗 128）单次前向并行算 5 个草稿位置的 base logits + 轻量 Markov head 建模草稿间依赖 + confidence head 预测逐位置接受概率——调度器结合引擎吞吐曲线动态选验证长度，最大化系统吞吐而非单请求速度。优化器三分法（8.2 节）：线性层 Muon、Q/K 用 head-wise Muon、embedding/Engram/预测头用动量 +Sinkhorn balancing。



图：DeepSeek-V4.1-Flash 总体架构

怎么看这张图：40 层明确分上下两段——下 20 层 causal encoder、上 20 层 decoder；注意 decoder 的全局 KV 从中段"横着"投影过去（CED），这是 prefill 减半的几何表达。底部三入口（Engram、文本、视觉 ViT）说明原生多模态；前 2 层仅 SWA，其余是各模式 CSA2。



图：CSA2 三种工作模式

怎么看这张图：三栏 Full / Reindex / Reuse，绿色=当前层计算量、黄色=复用的 main KV 与 indexer K、红色=复用的 Top-K 索引。从左到右绿色越来越少——Reuse 层几乎"免费"，这就是层维压缩省钱的来源。

演进位置：V4.1 是"压缩美学"的极致：三个可乘维度（条目、序列、层）× 位宽，一毫米一毫米地挤出 437 倍。它同时宣告 MTP 时代结束（DSpark 取代）与"记忆-计算解耦"（Engram）的开端。

再看一眼 40 层里注意力的完整排布：每层同时有全局注意力 + SWA 两个分支（前 2 层仅 SWA）；query 头 64、头维 512、query 压缩维 1280；indexer 32 头 × 128 维、top-k=512；输出投影 8 组 × 1024；SWA 窗口 128。MoE 全部层启用：1 shared + 384 routed（中间维 2304）、top-6，SwiGLU clamping 阈值 10；mHC 扩展因子 4，Sinkhorn-Knopp 20 次迭代；视觉侧 DeepSeek-ViT 从零训练（32 层、hidden 1024、16 头、patch 14、2D-RoPE）+ 3×3 pixel-unshuffle（视觉 token 数 ÷9）+ 2 层 MLP projector。预训练 45T 多模态 token 全程无 loss 不稳定——LR 台账：2000 步 warmup → 2.6e-4 恒定至 28T → cosine 降到 2.6e-5（40T）→ 恒定到 45T；batch 恒定 100.6M token；64K 长度从零起步训稀疏注意力，34T 时扩到 1M；Engram 学习率 ×5。

9.5 GLM-5（arXiv 2602.15763）：工程修补与异步 RL 基建

想解决什么问题：别人都在换架构，GLM-5 问的是另一个问题——**能不能用"工程修补"把现有 MLA 体系的短板补齐，同时把 RL 基建做成工业流水线，主打 agent 能力？** 副标题"from Vibe Coding to Agentic Engineering"点明定位。

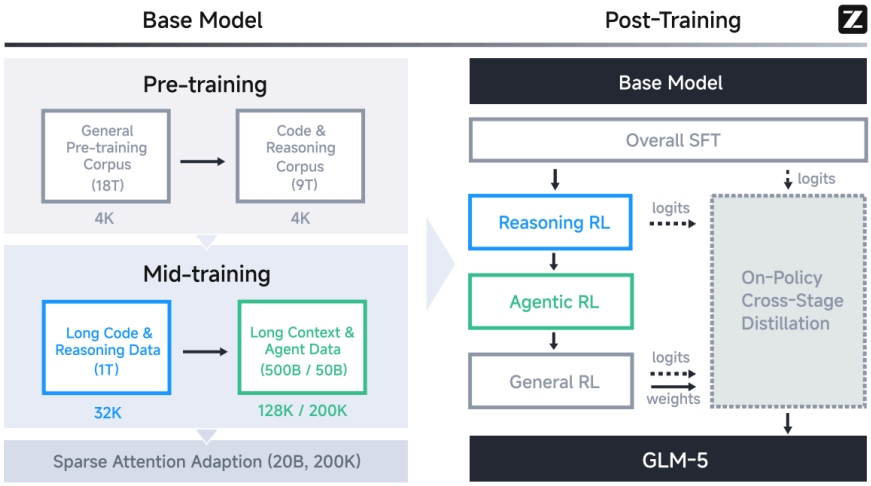
核心创意一句话：Muon Split 修好 MLA、DSA 低成本稀疏化降本、slime 异步框架 + 跨阶段蒸馏把 agentic RL 做成可规模化的生产线。744B 总参 / 40B 激活、80 层（故意减层加宽以降低 EP 通信）、256 专家、200K 上下文、28.5T token；Intelligence Index v4.0 得分 50（开源权重第一）。

创新点拆讲：

1. **Muon Split 与 MLA-256**（5.3 节）：发现并修复"MLA 在 Muon 下追不上 GQA-8"的隐蔽问题——上投影矩阵按注意力头拆小分别正交化即追平，训练全程 logits 尺度稳定无需 clipping；MLA-256（头维 192→256、头数减 1/3）训练计算与参数不变、decode 计算下降。
2. **DSA 低成本稀疏化**（5.4 节）：dense→sparse 两阶段续训只用 20B token（V3.2 用 943.7B），128K 下半价 GPU 成本；训练-推理一致性靠确定性 top-k 算子保证。9B 消融还探索了 search-based SWA 层配置（beam search 搜出 SFSSFFSS... 模式可泛化到 128K）与 SimpleGDN 线性化方案。
3. **slime 异步 agentic RL**（8.5 节）：server-based rollout（HTTP API 解耦）、尾延迟优化（EP64 + DP64 over 8 nodes、DP-attention for MLA 避免 KV 跨 rank 拷贝实现 no-queue serving）、FP8 rollout、MTP 投机解码（3 个 MTP 层共享参数，4 步投机下 accept length

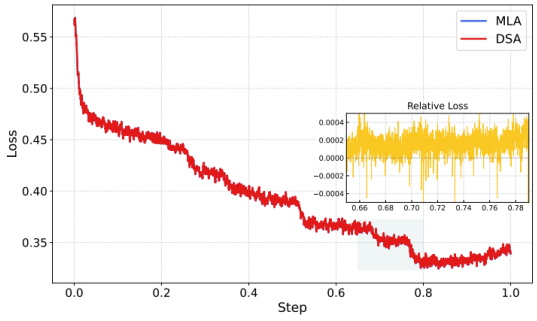
2.76，超 DeepSeek-V3.2 的 2.55）、心跳容错。数据侧约 1000 万 issue–PR 对、过滤后约 160B unique tokens，严格排除 AI 生成数据。

4. **General RL 与跨阶段蒸馏**：三维目标（基础正确性/情感智能/任务质量）+ 混合奖励（rule-based + ORM + GRM 互补）+ human-in-the-loop 风格锚点防"模型腔"；多阶段 RL 会退化先前能力，最终阶段用前序 checkpoint 做 on-policy 蒸馏恢复（8.5 节）。**为什么"模型腔"值得单独防**：RL 练久了模型会固化出某种高 reward 的腔调（句式模板化、过度礼貌、列表癖），人类读者一眼出戏。GLM-5 的办法是把人类专家的真实回复作为"风格锚点"混入奖励参考——奖励不再只问"答得对不对"，还问"像不像人写的"。这与 K3 的 Agentic GRM"超长度自动判负"是同一枚硬币的两面：**2026 年的奖励设计已经从"防答错"进化到"防油腻"**。



图：GLM-5 整体训练流水线

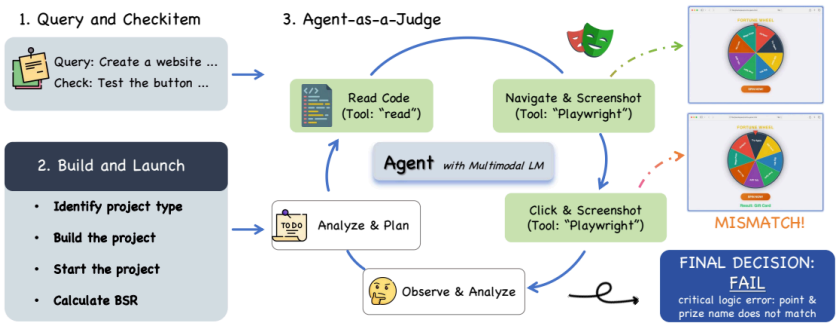
怎么看这张图：GLM-5 的"成长时间线"：预训练（18T+9T）→ 中期训练（长代码、长上下文 32K→200K）→ 稀疏注意力适配（DSA）→ 后训练四段（SFT → Reasoning RL → Agentic RL → General RL + 跨阶段蒸馏）。对照 R1 的四阶段图看，2026 年的后训练流程比 2025 年复杂了一个量级。



图：MLA 与 DSA 的 SFT loss 曲线对比

怎么看这张图：两条 loss 曲线几乎完全重合——"稀疏化无损"的直接证据，长上下文 90% 注意力条目冗余的"合法性证明"。

Success Rate (CSR) measures the fine-grained completion rate across all check-items. More details on the data distribution and the construction and validation process are in Appendix B.4.1.



图：Agent-as-a-Judge 评估流水线

怎么看这张图：前端项目评估流程：先静态构建验证，再由 Judge Agent 像真人一样逐项交互测试（读代码、Playwright 截图点击），最后判定功能正确性。它回答 agentic 时代的关键问题"奖励信号从哪来"——再派一个 agent 当考官。

演进位置：GLM-5 证明"不换架构也能前沿"：Muon Split、确定性 top-k 这类"补丁式"创新，加上工业化 RL 基建，同样能拿到开源第一。它贡献了本文最重要的一条工程哲学：**隐蔽的交互问题（优化器×架构、训练×推理）比显性的结构创新更影响成败**。

训练系统的另两处细节：交织式 PP 下的 **flexible MTP placement**——MTP 的 output 层与主 output 层同 stage 共享参数，embedding/transformer 放前一 stage，把各 stage 显存拉平；**Zero-redundant Muon 通信**——all-gather 仅限本 rank 拥有的参数分片、计算与通信重叠，消除峰值显存尖刺。上下文 mid-training 三阶段扩展：32K（1T token）→ 128K（500B）→ 200K（50B），越长的阶段数据量越小——长序列数据贵，好钢用在刀刃上。

9.6 Kimi K2.5（arXiv 2602.02276）：视觉与 agent 的正迁移

想解决什么问题：多模态模型通常"学了视觉掉文本"（能力此消彼长）；agent 任务靠单个模型串行执行又慢又受上下文限制。K2.5 问：**文本和视觉能不能互相增强？多个 agent 分身能不能并行干活、还越训越会分工？**

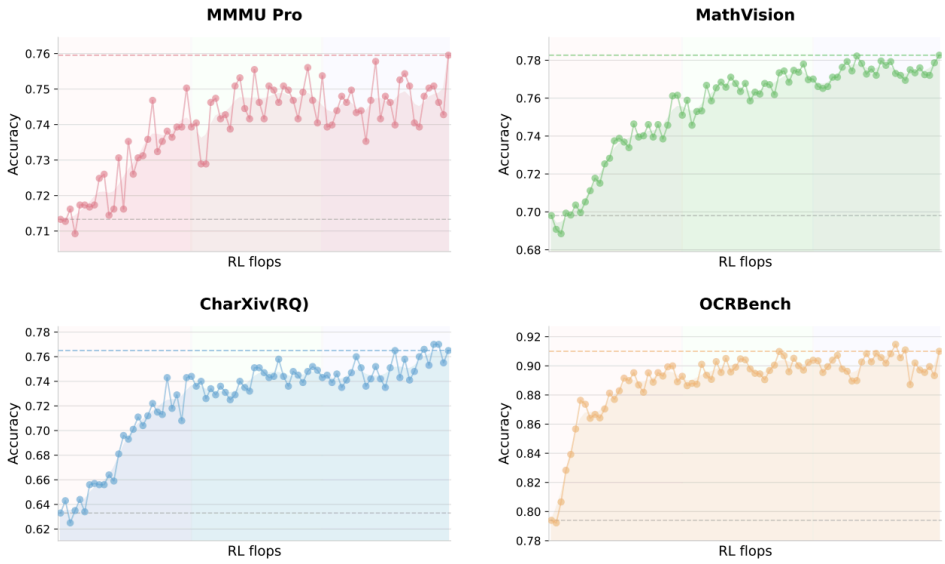
核心创意一句话：联合优化的三件套让文本与视觉双向增强；Agent Swarm + PARL 让模型学会"派分身并行干活"，把 test-time scaling 从"串行长推理"扩到"并行编排"。语言骨干沿用 K2（1.04T 总参 / 32B 激活、384 专家 top-8、MLA），续训约 15T 视觉-文本 token，YaRN 扩到 256K。

创新点拆讲：

1. **文本-视觉联合优化三件套**：①早期视觉融合 + 恒定混合比——固定 token 预算下，早期低比例混入视觉 token 优于晚期高比例（大白话：双语儿童从小混学两种语言，比成年后突击学第二外语扎实）；②**Zero-vision SFT**——纯文本 SFT 即可激活视觉推理与工具调用，加人工视

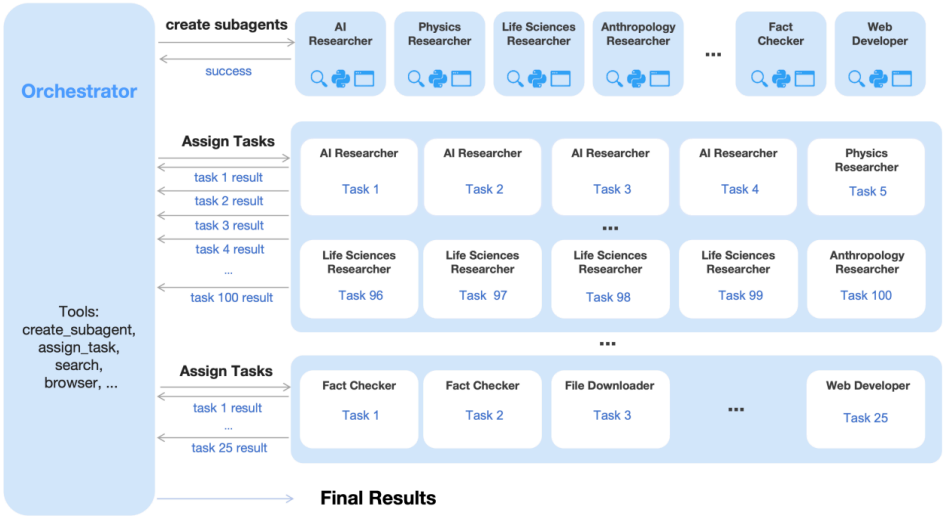
觉轨迹反而损害泛化（联合预训练已建好跨模态对齐，RL/SFT 只需"激活"）；③**联合文本-视觉 RL**——视觉 RL 反而提升 MMLU-Pro、GPQA-Diamond 等文本指标，双向增强。

2. **MoonViT-3D**：把 patch packing 推广到时间维——连续 4 帧组成时空体联合展平成 1D 序列，同一注意力同时处理空间与时间；MLP projector 前轻量时间池化 → **4 倍时间压缩**，同上下文窗口可处理 4 倍长视频；图像/视频编码器完全共享权重。
3. **Agent Swarm + PARL**（8.5 节）：orchestrator 可训练、子智能体冻结；三项奖励防串行塌缩；BrowseComp 78.4%（+17.8pt 超 GPT-5.2 Pro）、WideSearch 72.7→79.0、延迟降 3~4.5 倍且随任务复杂度近恒定。Agent Swarm 同时是"主动式上下文管理"：子任务语义隔离、只回传任务相关结果——"context sharding" 而非 truncation，等效扩展了有效上下文（BrowseComp 上优于 Discard-all 策略）。



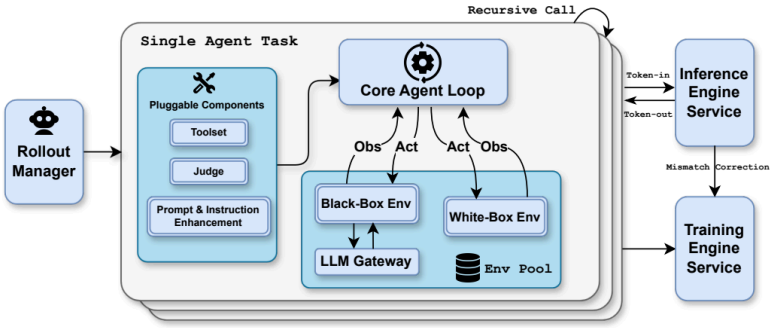
图：视觉 RL 训练曲线

怎么看这张图：四宫格是 MMMU Pro / MathVision / CharXiv / OCRBench 随 vision RL 计算量的曲线。起点是"极简 zero-vision SFT"——几乎没教过视觉推理的模型，纯靠 RL 拉起四条曲线，证明跨模态对齐在预训练已建好，RL 只需激活。



图：Agent Swarm 架构

怎么看这张图：中间可训练 orchestrator 把复杂任务（图中 100+ 项调研）分解成可并行子任务，动态创建冻结子智能体分头执行，只回传结果。子智能体之间互不通信——语义隔离既是上下文管理手段，也是训练稳定性来源。



图：K2.5 agentic RL 框架

怎么看这张图：三层结构：Rollout Manager 调度；单 agent 任务内是可插拔 Toolset（工具+沙箱）/ Judge（多面奖励）/ Core Agent Loop，接黑白盒环境与 Env Pool；底层推理与训练引擎服务。体会"可组合模块化"——环境与工具像积木一样替换。

演进位置：K2.5 的两个结论都反直觉且影响深远：多模态可以零和变双赢；test-time scaling 除了"想更久"还有"派更多人"。后者直接催生了 K3 的 9 专家分档 RL 体系。

【训练阶段台账】（供复现参考）：ViT 训练 1T token（seq 4096，纯 caption 交叉熵、无对比损失）→ 两阶段对齐（先对 Moonlight-16B-A3B 对齐 1T token，再短阶段只训 MLP projector 桥接 1T LLM）→ 联合预训练 15T（seq 4096）→ 联合长上下文 mid-training 500B + 200B（seq 32768 → 262144，YaRN 插值）。RL 基建支持**动态调整 subagent 与 orchestrator 的推理实例比例**以最大化集群利用率——这是"训练系统跟着算法变"的又一例。

Zero-vision SFT 再解释一句：它反直觉的地方在于"不教视觉却会视觉"。机理是联合预训练阶段视觉 token 与文本 token 已经在同一个表示空间里充分混合对齐——视觉推理的"能力"早已存在，SFT 的作用只是教模型"用指令的格式把能力调出来"，而格式是纯文本就能教的。人工视觉轨迹反而有害，因为手写轨迹的风格分布太窄，把模型从已对齐的表示空间带偏了。这个结论的实践意义很大：**多模态后训练可以复用纯文本的成熟管线**，成本大降。

K2.5 还有两个值得单独记住的 RL 细节：其一是 **token-efficient RL**——在保持性能的前提下显著压缩思考长度，缓解长 CoT 带来的显存与延迟压力（这也是对"思维链越长越好"直觉的一次修正：长度是手段不是目的，预算可控才是工程可部署的形态）；其二是 agentic RL 框架的**可组合模块化**设计——Toolset（工具+沙箱）、Judge（多面奖励信号）、Prompt 模块都可插拔替换，环境分黑盒/白盒两类并配 Env Pool 做实例池化。前者解决"agent 太贵"，后者解决"环境太难搭"——两者合起来才把 agentic RL 从 demo 推向生产线。

9.7 Kimi K3（arXiv 2607.35642）：三轴重构的 2.8T 旗舰

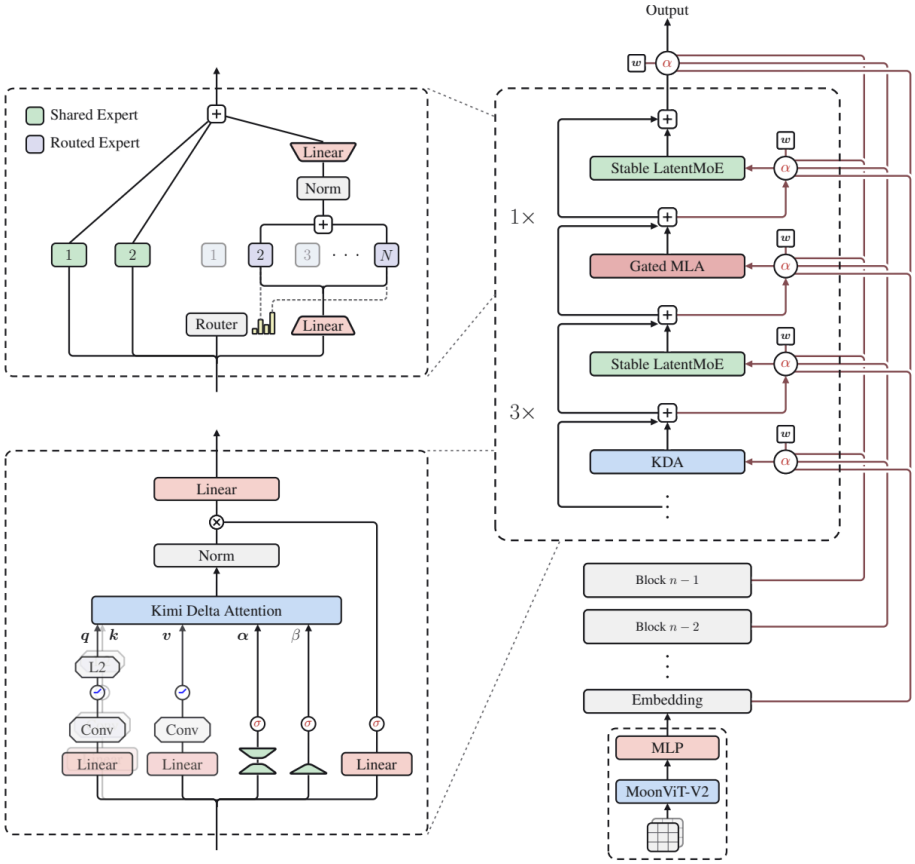
想解决什么问题：K2 体系（MLA + 384 专家）再往上堆参数，稀疏度和上下文都逼近极限——softmax 注意力的 cache 和 MoE 的通信都快撑不住。K3 问：**能不能沿序列、深度、宽度三个轴同时重构，把 scaling 效率直接提一个档次？**

核心创意一句话：序列轴换线性注意力为主（KDA）、深度轴把残差升级成跨层注意力（AttnRes）、宽度轴上 56 倍极端稀疏 MoE（Stable LatentMoE）——三轴齐换，scaling 效率提升约 **2.5 倍**。2.78T 总参 / 104.2B 激活、93 层、1M 上下文、160K 词表。

创新点拆讲：

1. **KDA + Gated MLA 混合**（5.5 节）：每 block 3 层 KDA + 1 层 Gated MLA，共 69 + 24 层。KDA 层零 KV cache；MLA 层全用 NoPE 免调 RoPE。系统配套：**FlashKDA**（CUTLASS chunkwise kernel，token 并行阶段与 head 并行递推独立调度）、**KCP**（跨设备上下文并行，转移矩阵精确复合 $S = \hat{S} + \prod M \cdot S_{in}$）、state-aware prefix caching（固定大小状态便于跨请求复用）。
2. **AttnRes（Attention Residuals）**：每层用学习到的 pseudo-query 对"embedding + 前序各 block 表示"做注意力，替代简单残差相加——大白话：普通残差像"每层楼直接把东西堆到上一层的仓库"，AttnRes 像"每层派人按需去历史各层的仓库里挑着取"。Block 变体把跨层注意力开销从 $O(Ld)$ 降到 $O(Nd)$（$N \approx 8$ 块），可用 online softmax 合并，推理期状态有界。
3. **Stable LatentMoE**（7.3 节）：896 专家 top-16、2 shared、稀疏度 56；LatentMoE 降维（3584）+ RMSNorm + SiTU-GLU + Quantile Balancing 四件套稳住极端稀疏。
4. **配套与后训练**：Per-Head Muon（8.2 节）；**MoonViT-V2 从零用 NTP 训练**（弃 SigLIP 对比初始化——梯度范数更低更稳、性能持平，证明对比预训练初始化非必需；2×2 pixel-shuffle 使 token 数 ÷4、支持 3584×3584）；QAT 从 SFT 起全程开启（MXFP4 权重 + MXFP8 激活）；后训练 3 域 × 3 reasoning effort = **9 个专家模型**，按（域, effort）选教师做逐 token OPD 合并（MOPD）；**partial rollout**（N×K 条轨迹完成 λ 比例即暂停开训，排队下轮优先

恢复，per-token 正则容忍陈旧数据）；**Reasoning Effort RL**（每题预算 $b_0(x)$、超 τb_0 奖励 −1，stage-wise 课程退火 τ 得到不同 effort 专家）；Agentic GRM 锦标赛式评估，超 $\sigma\ell_0$ 的啰嗦输出自动判负防 verbosity hacking。Scaling law 也重新搜索：cosine decay 在各自最优超参下一致优于 WSD。



图：Kimi K3 架构总览

怎么看这张图：围绕 token / channel / layer 三种 mixing 组织。找三个要点：每个 block 内"3 KDA + 1 Gated MLA"的节律；每个注意力层配对 Stable LatentMoE；左侧原生视觉通路（MoonViT-V2）。AttnRes 的 pseudo-query 连线显示每层都在"回头看"前序各 block。

演进位置：如果说 V3 证明"效率可以兑换能力"，K3 则证明"激进稀疏与线性化不只是省钱技巧，而是 scaling 的新引擎"——同样算力预算，K3 路线买到比 K2 路线多得多的智能。MoonEP、FlashKDA 等基建全部开源，让这条路线第一次有了可复现的公共底座。

K3 的 scaling law 重做也值得一提：换了架构（KDA 混合、AttnRes、LatentMoE）后，旧的 batch/学习率/形状配比全部失效，K3 重新搜索了 batch、LR、TPP（tokens per parameter）与模型形状；一个对全社区有用的结论是——**在各自最优超参下，cosine decay 一致优于 WSD 调度**（warmup 1%、wd 0.1）。这提醒我们：调度器之争没有脱离超参优化的"绝对赢家"，比较必须在各自调平之后进行。内存高效训练侧，K3 用统一激活管理抽象（重计算/量化/offload 都是可插拔存储策略，tensor 粒度注解），多数激活 block-wise FP8 量化 + offload/remote-offload，配合 SonicMoE 启发的 permuted-probs 梯度改写，把 2.78T 模型的训练显存压回可接受区间。

GEMM 调度与 shared expert 独立 stream 重叠。**一句话**：前面所有负载均衡技术决定"谁去哪个专家"，MoonEP 决定"到了之后怎么排得一张卡都不闲"——算法均衡与系统均衡终于闭环。

【小白 FAQ（第 9 章综合）】

- **Q：七篇论文里如果只精读一篇，选哪篇？** DeepSeek-V3。它的架构章节（MLA、MoE、FP8、DualPipe）是其余六篇的共同底座，且写得最系统。读完 V3 再读 R1（方法论）与 K3（新路线），其余按需。
- **Q：V4 为什么放弃纯 MLA？是 MLA 失败了吗？** 不是失败而是场景变了：MLA 压缩"每条 KV 的大小"，但条目数仍随上下文线性增长；1M 上下文下条目数本身成了瓶颈，必须叠加序列维压缩（CSA/HCA）与稀疏（DSA）。GLM-5 继续用 MLA-256 则说明在 200K 以内 MLA 仍然够用——**技术选型跟着上下文长度走**。
- **Q：各家都在做"多教师蒸馏合并"，会不会损失专家的峰值能力？** 论文们的答案是不会——关键在"on-policy"：教师在学生自己生成的轨迹上对齐，而不是让学生在陌生分布上模仿。V4 的全词表 logit 蒸馏还把估计方差压到最低。
- **Q：R1 的"纯 RL 涌现"能复现在小模型上吗？** R1 论文自己给了否定答案：32B 直接大规模 RL 显著不如蒸馏。涌现需要足够强的 base 能力作土壤——先把 base 训强，RL 才有东西可"挖"。

---

10. 技术演进总览：两年，三个战场

> 最后一章把镜头拉远：七篇论文不是七个孤立事件，而是三条清晰进化线上的里程碑。

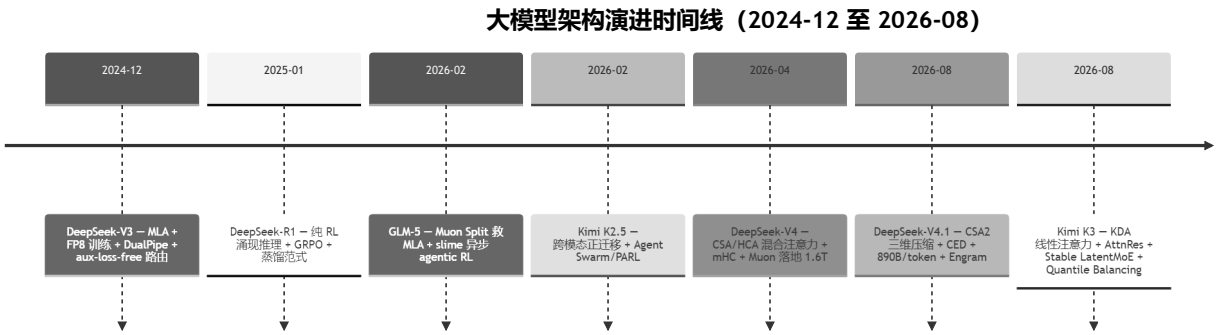
10.1 七模型规格对比

先教怎么读这张表：纵向是七个模型、横向是十条技术线。建议的阅读顺序是先扫"总参/激活"行感受规模膨胀（671B → 2.78T），再扫"KV cache"行感受压缩军备竞赛（576 元素 → 890 字节 → 25 层有 cache），最后按你关心的专题逐行对照。表里每一格都能在第 2~9 章找到对应的四层递进讲解。

维度	DeepSeek-V3	DeepSeek-R1	DeepSeek-V4 (Pro/Flash)	DeepSeek-V4.1-Flash	GLM-5	Kimi K2.5	Kimi K3
总参/激活	671B / 37B	671B / 37B	1.6T / 49B; 284B / 13B	552B + 196B Engram / 16B decode、8B prefill	744B / 40B	1.04T / 32B	2.78T / 104.2B

层数 / hidden	61 / 7168	61 / 7168	61 / 7168；43 / 4096	40（20+20）/ 5120	80 / —	61 / 7168	93 / 7168
Attention	MLA（128 头，d_c=512）	同 V3	CSA(4×)+HCA(128×) 交错	CSA2 跨层复用 + CED	MLA-256 + DSA	MLA	69 KDA + 24 Gated MLA(NoPE)
KV cache	576 元素/token/层	同 V3	1M 下为 V3.2 的 1/9.5	**890 B/token** 全局	latent + top-k 2048	同 V3 系	仅 25 层有压缩 cache
MoE	1S+256R top-8	同 V3	1S+384R/256R top-6	1S+384R top-6，模态双偏置	256 专家	384R top-8 + 1S	896R top-16 + 2S，LatentMoE
路由均衡	aux-loss-free 偏置	同 V3	同 V3（去节点限制）	+ 图文双套偏置	aux-loss-free	aux-loss-free	**Quantile Balancing**
优化器	AdamW	AdamW	**Muon**（hybrid NS）	Muon + head-wise + Sinkhorn	**Muon + Muon Split**	**MuonClip + QK-Clip**	**Per-Head Muon**
量化	FP8 混合精度训练	沿用 V3	KV FP8 + indexer FP4 + FP4 QAT	FP4 主 KV cache	FP8 rollout	—	MXFP4 权重 + MXFP8 激活 QAT
后训练	SFT + GRPO + R1 蒸馏	纯 RL 涌现 + 四阶段	专家 RL → 多教师 OPD	SFT→GRPO →OPD + DSpark	异步 agentic RL + 跨阶段蒸馏	联合文本-视觉 RL + PARL	9 专家 RL → MOPD
上下文	128K（YaRN）	32K rollout	**1M**	**1M**	200K	256K	**1M**
预训练量	14.8T	基于 V3	32T / 33T	45T 多模态	28.5T	15T（联合）	未披露

10.2 三条演进主线



主线一：Attention——三家同向而不同路。 共同结论是"1M 上下文中绝大多数 token 交互是冗余的"，但解法各异：DeepSeek 走"softmax 注意力的 KV 压缩 + 稀疏"（MLA → DSA → CSA/HCA → CSA2），把条目大小、条目数、层数、位宽四个因子挨个砍；Kimi 走"线性注意力为主 + 周期

全局注意力"（KDA + Gated MLA），从根上取消增长的 KV cache；GLM 走"MLA 工程修补 + DSA 借用"（Muon Split、MLA-256），以最小改动拿稀疏收益。

主线二：负载均衡——从 loss 到偏置再到分位数。 aux-loss（污染主任务）→ aux-loss-free 固定步长偏置（V3，温和但会振荡）→ 模态/历史参数扩展（V4.1 双套偏置、V4 Anticipatory Routing）→ Quantile Balancing 一步精确匹配（K3）。一条"让均衡越来越精确、对主任务越来越无感"的进化链。

主线三：后训练——收敛到同一范式。 R1 在 2025 年初确立"RL 可以涌现推理"；到 2026 年三家不约而同走向"先分领域/分推理档位训多个专家（GRPO），再多教师 on-policy 蒸馏合并"（V4 的 OPD、K3 的 MOPD、GLM-5 跨阶段蒸馏），同时把 RL 基建做成异步、容错、百万 token 级系统工程（slime、DSec 沙箱、partial rollout、microVM）。**后训练从"炼丹技巧"变成"工业流水线"。**

还有一条暗线值得点名：确定性。 V4 的 batch-invariant kernel、GLM-5 的确定性 top-k、K3 的静态形状 MoonEP，看似零散，实则同一主题——**当训练规模到几十万卡时，"两次运行结果一致"从奢侈品变成必需品**：没有它，loss spike 无法归因、训练/推理一致性无法保证、agent 环境的 reward 无法复现。确定性是超大规模训练时代的"工程质量地基"。

10.3 给读者的三点总结

1. **效率即能力。** 没有一项效率优化是"纯省钱"：MLA 省下的 KV cache 让 R1 的上万 token 思维链 RL 变得可负担；FP8 省下的显存让 671B 塞进 2048 卡集群；KDA 取消 KV cache 让 1M 上下文 agentic rollout 成为可能。每次效率突破都直接兑换成能力上限的抬升。
2. **架构与系统是同一枚硬币。** Muon 遇 ZeRO 要解分桶、压缩注意力遇 CP 要重设计通信、MoE 遇 EP 要重写 all-to-all——2026 年架构论文近半篇幅在讲系统工程，因为**一个新结构只有配上为它定制的系统才真正存在**。
3. **下一代战场已经清晰。** 注意力继续向"近乎免费的全局交互"逼近（稀疏与线性两条路线未分胜负）；负载均衡追求"零感知"精确调度；后训练把环境合成、agent 沙箱、蒸馏合并做成标准产线。读懂这七篇论文，就拿到了观察下一轮竞赛的入场券。

【小白 FAQ（全局收尾）】

- **Q：读完本文，下一步怎么动手？** 三条路：想跑模型就从 Qwen/Llama 的开源小模型 + vLLM 开始体验 KV cache 与 PD 分离；想读懂论文就从 DeepSeek-V3 原文配本文第 9.1 节开始；想做研究就把第 10 章三条主线各找一篇最新跟进工作读。无论哪条路，都建议动手"复算"本文里的数值例子——能独立推出 890 B/token 的人，讨论长上下文架构时就有底气。
- **Q：这些技术会不会很快过时？** 具体的系数会过时，但"瓶颈分析 → 定位因子 → 砍因子"的方法论不会。第 2 章的三瓶颈框架在十年前适用、十年后大概率仍适用——变的只是哪个因子当下最贵。

- **Q：为什么没有讲多模态生成的图像/视频模型？** 本文聚焦语言模型的架构与训练主干；V4.1/K2.5/K3 的视觉编码器（DeepSeek-ViT、MoonViT 系列）属于"理解侧"输入通路，生成侧（扩散模型等）是另一套技术栈，值得另写一篇。

延伸阅读

以下七篇论文是本文全部素材来源，按出场顺序列出，建议配合各论文架构图原文阅读：

1. **DeepSeek-V3 Technical Report**. DeepSeek-AI, arXiv:2412.19437, 2025-02（初版 2024-12）. MLA + DeepSeekMoE + FP8 训练 + DualPipe 的奠基之作。
2. **DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning**. DeepSeek-AI, arXiv:2501.12948（Nature 版本 2026-01）. 纯 RL 涌现推理与四阶段后训练流水线。
3. **DeepSeek-V4 Technical Report: Towards Highly Efficient Million-Token Context Intelligence**. DeepSeek-AI, arXiv:2604.19348, 2026-04（preview）. CSA/HCA 混合注意力、mHC、Muon 与多教师 OPD。
4. **DeepSeek-V4.1-Flash Technical Report: Pushing the Limits of KV Cache Compression**. DeepSeek-AI, 2026-08. CSA2 跨层复用、CED、FP4 主 KV cache 与 890 B/token。
5. **GLM-5: from Vibe Coding to Agentic Engineering**. ZhipuAI & Tsinghua University, arXiv:2602.15763, 2026-02. Muon Split / MLA-256、DSA 续训与 slime 异步 agentic RL。
6. **Kimi K2.5: Visual Agentic Intelligence**. Moonshot AI, arXiv:2602.02276, 2026-02. 跨模态正迁移三件套与 Agent Swarm / PARL。
7. **Kimi K3: Open Frontier Intelligence**. Moonshot AI, arXiv:2607.35642, 2026-08. KDA 线性注意力、AttnRes、Stable LatentMoE 与 Quantile Balancing。

进一步阅读建议：想深入注意力数学细节可从 DeepSeek-V2（MLA 首次提出）与 NSA 论文入手；想深入并行系统可读 Megatron-LM、ZeRO 与 vLLM（PagedAttention）原文；想深入 RL 可从 PPO 原文与 DeepSeekMath（GRPO 首次提出）读起。读完这七篇再加这五篇，你就拥有了覆盖当前大模型技术全景的最小知识集。

致谢式说明：本文全部技术数字均摘自上述论文的技术笔记（workspace 内 notes/ 目录），插图提取自论文 PDF 原文（figures/ 目录，渲染精度约 216 DPI）。若文中数字与你手中的论文版本有出入，以论文最新版本为准——预印本时代的论文会持续修订，读原文时顺手核对版本号与发布日期，是避免引用过期数据的好习惯。